IBM

AS/400™

SC21-9659-0

Programming:
Data Base Guide

IBM

AS/400™

SC21-9659-0

Programming: Data Base Guide

First Edition (June 1988)

This edition applies to Release 1, Modification Level 0, of the IBM Operating System/400 Licensed Program (Program 5728-SS1), and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or technical newsletters.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

The numbers at the bottom right of illustrations are publishing control numbers and are not part of the technical content of this manual.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to your IBM-approved remarketer.

This publication could contain technical inaccuracies or typographical errors. A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department 245, Rochester, Minnesota, U.S.A. 55901. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

AS/400 is a trademark of the International Business Machines Corporation.

© Copyright International Business Machines Corporation 1988

About This Guide

This guide contains information about the AS/400 data base management system. The guide describes how to set up and use a data base on the AS/400 system.

This guide does not cover in detail all the data base related capabilities on the AS/400 system. The *Programming: Structured Query Language/400 Programmer's Guide* (to be available at a later date) and *Programming: Structured Query Language/400 Reference* (to be available at a later date) describe how to use Structured Query Language/400 in more detail. The *Programming: Data Description Specifications Reference*, SC21-9620 describes how to specify OS/400 DDS (data description specifications). The *Programming: Control Language Reference*, SBOF-0481 describes how to specify OS/400 control language (CL) commands. The *Utilities: Interactive Data Definition Utility User's Guide*, SC21-9657 describes the OS/400 IDDU (interactive data definition utility) and data dictionary in more detail. The *Programming: Backup and Recovery Guide*, SC21-8079 describes data base recovery considerations in more detail.

Who Should Use This Guide

This guide is intended for the system administrator or programmer who creates and manages files and data bases on the system. In addition, the guide is intended for programmers who use the data base in their programs.

What You Should Know

Before using this guide, you should be familiar with the introductory material for using the system. You should also understand how to write a high-level language program for the AS/400 system.

If you plan to use Structured Query Language/400 to create and process your data base files, see the appropriate guides in "Related Printed Information" on page v.

If you plan to use OS/400 IDDU to define data on the system, see the appropriate guides in "Related Printed Information" on page v.

If you plan to use the data base recovery functions on the AS/400 system, you should be familiar with the topics discussed in the *Programming: Backup and Recovery Guide*, SC21-8079.

How This Guide Is Organized

The parts in this guide are organized as follows:

- Introducing the AS/400 data base

  The chapters in this part describe the concepts of data base systems and how the data base is implemented on the AS/400 system. This part also describes the basic steps in setting up, processing, and managing data base files on the AS/400 system.

- Setting up data base files

  The chapters in this part of the guide describe in detail how to define and create data base files on the system.

- Processing data base files in programs

  The chapters in this part of the guide describe in detail how to process data base files in programs. Included is a description of opening data base files, reading, updating, adding, and deleting records in the file, and closing data base files.

- Managing data base files

  This part of the guide describes additional functions and considerations when using the data base on the AS/400 system. Included is a description of managing data base file members, using data base cross-reference information, and planning considerations for data base recovery.

- Appendixes

  The appendixes discuss the minimum and maximum file sizes on the AS/400 system, the double-byte character set (DBCS) considerations, the data base lock considerations, and the output files for the Office-related control language (CL) commands.

- Glossary

  The glossary defines words used in this guide. Use the glossary if you find an unfamiliar word.

- Index

  The index lists data base processing tasks used in this guide. Use the index when you want to find instructions for a specific task.

Related Online Information

The following online information is available on the AS/400 system. You can press the Help key a second time to see an explanation of how the online information works, including the index search function.

Help for Displays

You can press the Help key on any display to see information about the display. There are two types of help available:

General
Specific

General help explains the purpose of the display. General help appears if you press the Help key when the cursor is outside the area for which specific help is available.

Specific help explains the field on which the cursor is positioned when you press the Help key. For example, it describes the choices available for a prompt. If a system message appears at the bottom of the display, position the cursor on the message and press the Help key to see the information about the cause of the message and the appropriate action to take.

To exit online information, press F3 (Exit). You return to the display on which you pressed the Help key.

Index Search

Index search allows you to specify the words or phrases you want to see information about. To use index search, press the Help key, then press F11 (Search index).

Help for Control Language Commands

To see prompts for parameters for a control language command, type the command, then press F4 (Prompt) instead of the Enter key.

Online Education

AS/400 online education provides tutorials on a wide variety of topics. To use the online education, press F13 (User support) on any system menu to show the User Support menu. Then select the option to use online education.

Question-and-Answer Function

The question-and-answer (Q & A) function provides answers to questions you may have about using the AS/400 system. To use the Q & A function, press F13 (User support) on any system menu to show the User Support menu. Then select the option to use the question-and-answer function.

Related Printed Information

If you need more information, refer to the following guides:

- *Query: User's Guide*, SC21-9614. This guide has information about using AS/400 Query.
- *Communications: Distributed Data Management User's Guide*, SC21-9600. This guide has information about using OS/400 DDM (distributed data management).
- *Programming: Backup and Recovery Guide*, SC21-8079. This guide has information about using the backup and recovery functions of the AS/400 system.
- *Programming: Control Language Programmer's Guide*, SC21-8077. This guide has information about using the command language on the AS/400 system.
- *Programming: Control Language Reference*, SBOF-0481. This set of manuals has detailed information about all system commands, including data base related commands.

  This set of manuals is divided into five volumes. Volume 1 provides an overview of the control language commands and describes the syntax coding rules needed to code them. Volumes 2 through 5 describe every control language command, including commands for the system control program, languages, and utilities. The order numbers for these volumes are:

  Volume 1 SC21-9775
  Volume 2 SC21-9776
  Volume 3 SC21-9777
  Volume 4 SC21-9778
  Volume 5 SC21-9779

- *Programming: Data Description Specifications Reference*, SC21-9620. This guide has detailed information about using OS/400 DDS (data description specifications).

- *Programming: Data Management Guide*, SC21-9658. This guide has information about using other data management tools on the AS/400 system. Included are topics on the Copy File (CPYF) command and the override commands.
- *Programming: Security Concepts and Planning*, SC21-8083. This guide has information about using the integrated security features of the AS/400 system to secure your data base file information.
- *Programming: Structured Query Language/400 Reference* (to be available at a later date). This guide has detailed information about Structured Query Language/400 statements.
- *Programming: Structured Query Language/400 Programmer's Guide* (to be available at a later date). This guide has information about using SQL/400.
- *Utilities: Interactive Data Definition Utility User's Guide*, SC21-9657. This guide has information about using OS/400 IDDU (interactive data definition utility).

Contents

Part 1. Introducing the AS/400 Data Base

Chapter 1. The AS/400 Data Base 1-1
Data Base Concepts 1-1
The Integrated AS/400 Data Base Management System 1-4

Chapter 2. The AS/400 Data Base Structure 2-1
Fields and Field Definitions 2-1
Records and Record Formats 2-1
Data Base Files and File Descriptions 2-1
Physical Files 2-1
Logical Files 2-2
Data and Source Files 2-2
Members 2-3
Access Paths 2-4
Arrival Sequence Access Path 2-4
Keyed Sequence Access Path 2-4

Chapter 3. Working with the AS/400 Data Base 3-1
Setting Up a Data Base File 3-1
Describing the Data Base File 3-1
Creating the Data Base File 3-2
An Example of Using DDS to Set Up Data Base Files 3-2
Processing a Data Base File in a Program 3-3
Managing Data Base Files 3-3
Member Operations 3-4
Data Base Cross-Reference Information 3-4
Data Base Recovery 3-4
Remote File Access 3-5

Part 2. Setting Up Data Base Files

Chapter 4. General Considerations 4-1
Describing Data Base Files 4-1
Dictionary-Described Data 4-2
Methods of Describing Data to the System 4-3
Describing a Data Base File to the System 4-4
Describing Data Base Files Using DDS 4-6
Describing the Access Path for the File 4-16
Creating Data Base Files 4-23
Specifying Data Base File and Member Attributes 4-23
File and Member Name 4-23
Source File and Member 4-24
Type of Data Base File 4-24
Number of Members Allowed in the File 4-24
Where the Data Is to Be Stored 4-24
Frequency of Writing Data to Auxiliary Storage 4-24
Frequency of Writing the Access Path to Auxiliary Storage 4-25
Checking for Changes to the Record Format Description 4-25

Keeping the Access Path Current 4-25
Recovering an Access Path If the System Fails 4-27
Sharing a File in the Same Job 4-28
Specifying Wait Time for a Locked File or Record 4-28
Specifying Public Authority 4-28
Specifying The System on Which the File Is Created 4-29
Specifying File and Member Text 4-29

Chapter 5. Setting Up Physical Files 5-1
Creating a Physical File 5-1
Specifying Physical File and Member Attributes 5-1
Expiration Date 5-1
Size of the Physical File Member 5-2
Storage Allocation 5-2
Method of Allocating Storage 5-2
Record Length 5-3
Deleted Records 5-3
Physical File Capabilities 5-3

Chapter 6. Setting Up Logical Files 6-1
Describing Logical File Record Formats 6-1
Describing Field Usage for Logical Files 6-3
Deriving New Fields from Existing Fields 6-4
Describing Floating-Point Fields in Logical Files 6-7
Describing Access Paths for Logical Files 6-8
Selecting and Omitting Records Using Logical Files 6-8
Using Existing Access Paths 6-13
Creating a Logical File 6-14
Creating a Logical File with More Than One Record Format 6-15
Logical File Members 6-19
Join Logical File Considerations 6-21
Basic Concepts of Joining Two Physical Files (Example 1) 6-22
Setting Up a Join Logical File 6-30
Using More Than One Field to Join Files (Example 2) 6-31
Reading Duplicate Records in Secondary Files (Example 3) 6-33
Using Join Fields Whose Attributes Are Different (Example 4) 6-34
Describing Fields That Never Appear in the Record Format (Example 5) . 6-36
Specifying Key Fields in Join Logical Files (Example 6) 6-38
Specifying Select/Omit Statements in Join Logical Files 6-39
Joining Three or More Physical Files (Example 7) 6-39
Joining a Physical File to Itself (Example 8) 6-41
Using Default Data for Missing Records from Secondary Files (Example 9) 6-43
A Complex Join Logical File (Example 10) 6-45
Performance Considerations 6-46
Data Integrity Considerations 6-47
Summary of Rules for Join Logical Files 6-47

Chapter 7. Data Base Security 7-1
File and Data Authority 7-1
Object Operational Authority 7-1
Object Existence Authority 7-1
Object Management Authority 7-2
Data Authorities 7-2
Public Authority 7-3
Data Base File Capabilities 7-4
Using Logical Files to Secure Data 7-4

Part 3. Processing Data Base Files in Programs

Chapter 8. Run Time Considerations 8-1
File and Member Name 8-2
File Processing Options 8-2
Specifying the Type of Processing 8-2
Specifying the Initial File Position 8-3
Ignoring the Keyed Sequence Access Path 8-3
Delaying End of File Processing 8-3
Specifying the Record Length 8-3
Ignoring Record Formats 8-4
Data Recovery and Integrity 8-4
Protecting Your File with the Journal and Commitment Control 8-4
Writing Data and Access Paths to Auxiliary Storage 8-4
Checking Changes to the Record Format Description 8-4
Checking for the File's Expiration Date 8-5
Preventing the Job from Changing Data in the File 8-5
Sharing Data Base Files Across Jobs 8-5
Record Locks 8-5
File Locks 8-6
Member Locks 8-6
Record Format Data Locks 8-7
Sharing Data Base Files in the Same Job 8-7
Open Considerations for Files Shared in a Job 8-7
Input/Output Considerations for Files Shared in a Job 8-8
Close Considerations for Files Shared in a Job 8-9
Sequential-Only Processing 8-13
Open Considerations for Sequential-Only Processing 8-13
Input/Output Considerations for Sequential-Only Processing 8-14
Close Considerations for Sequential-Only Processing 8-15
Run Time Summary 8-15

Chapter 9. Opening a Data Base File 9-1
Opening a Data Base File Member 9-1
Using the Open Data Base File (OPNDBF) Command 9-1
Using the Open Query File (OPNQRYF) Command 9-2
Using an Existing Record Format in the File 9-3
Using a File with a Different Record Format 9-5
Selecting Records without Using DDS 9-8
Specifying a Keyed Access Path without Using DDS 9-15
Specifying Key Fields from Different Files 9-16
Dynamically Joining Data Base Files without DDS 9-16
Handling Missing Records in Secondary Join Files 9-20
Unique-Key Processing 9-21
Defining Fields Derived from Existing Field Definitions 9-22
Handling Divide by Zero 9-24
Summarizing Data from Data Base File Records (Grouping) 9-25
Final Total-Only Processing 9-28
Controlling How the System Runs the Open Query File Command 9-29
Considerations for Creating a File and Using the FORMAT Parameter . . 9-30
Considerations for Arranging Records 9-31
Considerations for DDM Files 9-31
Considerations for Writing a High-Level Language Program 9-31
Messages Sent When the Open Query File (OPNQRYF) Command Is Run 9-32

Using the Open Query File (OPNQRYF) Command for More Than Just Input 9-32
Using the Open Query File (QPNQRYF) Command for Random Processing 9-33
Performance Considerations 9-33
Performance Comparisons with Other Data Base Functions 9-34
Considerations for Field Use 9-35
Considerations for Files Shared in a Job 9-36
Considerations for Checking If the Record Format Description Changed . 9-36
Other Run Time Considerations 9-36
Typical Errors When Using the Open Query File (OPNQRYF) Command 9-37

Chapter 10. Basic Data Base File Operations 10-1
Setting a Position in the File 10-1
Reading Data Base Records 10-2
Reading Data Base Records Using an Arrival Sequence Access Path 10-2
Reading Data Base Records Using a Keyed Sequence Access Path 10-3
Waiting for More Records When End of File Is Reached 10-4
Releasing Locked Records 10-6
Updating Data Base Records 10-6
Adding Data Base Records 10-7
Identifying Which Record Format to Add in a File with Multiple Formats 10-7
Using the Force-End-Of-Data Operation 10-9
Deleting Data Base Records 10-9

Chapter 11. Closing a Data Base File 11-1

Chapter 12. Handling Data Base File Errors in a Program 12-1

Part 4. Managing Data Base Files

Chapter 13. Managing Data Base Members 13-1
Member Operations Common to All Data Base Files 13-1
Adding Members to Files 13-1
Changing Member Attributes 13-2
Renaming Members 13-2
Removing Members from Files 13-2
Physical File Member Operations 13-2
Initializing Data in a Physical File Member 13-2
Clearing Data from Physical File Members 13-3
Reorganizing Data in Physical File Members 13-3
Displaying Records in a Physical File Member 13-5

Chapter 14. Changing Data Base File Descriptions and Attributes 14-1
Effect of Changing Fields in a File Description 14-1
Changing a Physical File Description and Attributes 14-2
Changing a Logical File Description and Attributes 14-5

Chapter 15. Using Data Base Attribute and Cross-Reference Information . . . 15-1
Displaying Information about Data Base Files 15-1
Displaying Attributes for a File 15-1
Displaying the Descriptions of the Fields in a File 15-1
Displaying the Relationships between Files on the System 15-2
Displaying the Files Used by Programs 15-2
Displaying the System Cross-Reference Files 15-4
Writing the Output from a Command Directly to a Data Base File 15-4

Chapter 16. Planning Data Base Recovery 16-1
Data Base Save and Restore Considerations 16-1
Save and Restore Commands 16-1
Saving and Restoring Data Base File Members 16-1
Saving and Restoring Access Paths 16-1
Saving and Restoring Data Using a Disk-Resident Save File 16-2
Data Base Data Recovery 16-2
Journal Management 16-2
Writing Data to Auxiliary Storage 16-3
Transaction Recovery 16-3
Access Path Recovery 16-5
Rebuilding Access Paths 16-5
Journaling Access Paths 16-5
Other Methods to Avoid Rebuilding Access Paths 16-6
Controlling When Access Paths Are Rebuilt 16-7
Designing Files to Reduce Access Path Rebuilding Time 16-8
Data Base Recovery after an Abnormal System End 16-8
Data Base File Recovery during the IPL 16-8
Data Base File Recovery after the IPL 16-9
Data Base File Recovery Options Table 16-10

Chapter 17. Using Source Files 17-1
Source File Concepts 17-1
Creating a Source File 17-1
IBM-Supplied Source Files 17-2
Source File Attributes 17-2
Creating Source File without DDS 17-3
Creating Source Files with DDS 17-3
Working with Source Files 17-3
Using the Source Entry Utility 17-3
Using Device Source Files 17-4
Copying Source File Data 17-4
Using Source Files in a Program 17-5
Creating an Object Using a Source File 17-6
Creating an Object from Source Statements in a Batch Job 17-6
Determining Which Source File Member Was Used to Create an Object . . 17-7
Managing a Source File 17-8
Changing a Source File's Attributes 17-8
Reorganizing Source File Member Data 17-8
Determining When a Source Statement Was Changed 17-9
Considerations for Saving Source Files 17-9
Using Source Files for Documentation 17-10

Appendix A. Data Base File Sizes A-1

Appendix B. Double-Byte Character Set (DBCS) Considerations B-1
DBCS Field Data Types B-1
DBCS Field Mapping Considerations B-1
DBCS Field Concatenation B-2
Using DBCS Fields in the Open Query File (OPNQRYF) Command B-2
Comparing DBCS Fields B-2
Using the Wild Card Function with DBCS Fields B-2

Appendix C. Data Base Lock Considerations C-1

Appendix D. Office Output File Considerations D-1

Display Directory (DSPDIR) Outfile Considerations D-1
Display Distribution List (DSPDSTL) Outfile Considerations D-3
Display Folder (DSPFLR) Outfile Considerations D-4
Folder Object Information D-4
Document Object Information D-5
File Document (FILDOC) File Considerations D-6
Print Document (PRTDOC) Outfile Considerations D-6
File Format for PRTFILE Files D-7
Query Document Library (QRYDOCLIB) Outfile Considerations D-11
Query Distribution (QRYDST) Outfile Considerations D-16
Incoming Distribution Information D-17
Outgoing Distribution Information D-19
Receive Distribution (RCVDST) Outfile Considerations D-22
Replace Document (RPLDOC) File Considerations D-27
Retrieve Document (RTVDOC) Outfile Considerations D-28
Send Distribution (SNDDST) File Considerations D-34

Glossary G-1

Index X-1

Part 1. Introducing the AS/400 Data Base

Chapter 1. The AS/400 Data Base

This chapter introduces the AS/400[1] data base. It discusses why you might want to use a data base management system instead of a system with no data base manager. The chapter also discusses how the data base is implemented on the AS/400 system.

Data Base Concepts

The term data base is often associated with complex operating environments and large data processing staffs. In fact, a **data base** is simply a place to store your data. In this sense, the files on your current system can be viewed as a data base. What sets a system with a **data base management system** apart from one without a data base management system is that a data base management system helps manage the data in the data base for you, rather than you doing all of the work.

For example, if you need to read data for customer number 100 and that customer has information in both a master file record and an address file record, you would normally do two read operations in your program (one read for each file). However, if you were using a data base manager, your program could do one read operation for customer number 100, and the data base manager could get the data from the two files and present it to your program as if it were coming from a single file. Your program does not need to know how many files, and from which files, the data is being read.

Another example of how a data base management system can help you concerns adding new fields to a file. Most programs on systems without a data base are tied very closely to the way data is physically stored on the system. That is, if you have defined a file as containing fields *A*, *B*, and *C*, each program that uses that file is coded with the knowledge of that record layout. If you decide later to add field *D*, all your programs that use that file must be changed and recompiled.

A data base management system, on the other hand, can hide the physical record layout of a file. In the example above, the data base system could present data to your program as if the record layout was still *A*, *B*, *C* (even though the new physical record layout was *A*, *B*, *C*, *D*). Your programs would not have to change or be recompiled (unless, of course, the programs needed to use the new field).

Data base management systems are often characterized by the following attributes:

- Data and program independence
- Data sharing across applications
- Data secured at the file, record, and field levels
- Data integrity and recovery
- Data joined from more than one file
- Data selected and arranged based on processing requirements

In addition, data base management systems are often associated with a data definition function. This function allows you to define data one time to the system, and that definition is maintained by the system.

---

1 AS/400 is a trademark of the International Business Machines Corporation.

Data and Program Independence

The AS/400 data base management system separates the program's logical view of the data from the way the data is physically stored on the system. The ability to logically view data independent of its physical structure, allows you to make changes to the physical data without, necessarily, having to change or recompile your programs. For example, you can add fields to a file, remove fields from a file, or change the length of fields in a file without changing anything in programs that do not use those fields.

Data Sharing

The AS/400 data base management system allows many users and programs to share concurrently the data in the data base files. The AS/400 system maintains the integrity of the data regardless of the number of programs sharing the data or how those programs use the data. In addition, the system ensures that changes to data base records are reflected in the data base. Therefore, programs see the latest information. New programs and applications can be added to the system and have immediate access to the shared data, without affecting existing programs.

Because data base management systems allow greater data sharing, you often can reduce the amount of data that you must define and store. Traditionally, each application has its own files. For example, both the inventory application and the accounts payable application might have their own version of the inventory file. The inventory application file might contain an inventory item number field, an item description field, and a warehouse location field. The accounts payable file might have an inventory item number field, an item description field, and a supplier number field.

The AS/400 data base management system shares data across applications, reducing the need to duplicate data. In the previous example, one file containing the inventory item number field and item description field could be created and shared by both the accounts payable and inventory applications.

In addition, the data base uses record locks to ensure data integrity and the ability to safely share data in the data base. That is, the system automatically protects a record from being updated by more than one user at the same time. The system can also ensure that incomplete transactions are not seen by other users until a transaction is complete.

Data Security

Because the AS/400 data base management system allows you to share data easily across many applications and users, you need a way to prevent unauthorized use of the data. The AS/400 system supplies a comprehensive security function to allow you to secure your data at many levels. For example, you have defined FILEA as containing three fields: customer name, customer address, and amount due. You can:

- Give authority for the whole file to specified users. For example, Cheryl and Lois may access FILEA, but Rick cannot.
- Give authority for certain types of file operations to specified users. For example, Dave may read, but not update any records in FILEA.
- Give authority for certain fields to be read by specified users. For example, Mark's view of the file includes the customer name and customer address fields, but not the amount due field.
- Give authority for certain fields to be changed by specified users. For example, Robin's view of the file may include all the fields, but Robin may not change the amount due field.

- Give authority for certain records to specified users. For example, records with a certain value can only be viewed by the payroll department.

The ability to secure data at the file, record, and field levels may be combined in different ways for different users, according to your business requirements.

Data Integrity

The system allows you to journal (record) all changes that occur to the data base. The journal can then be used for recovering your data in case the data base file must be reset to a previous point.

The AS/400 data base management system can recover data to the transaction level. That is, if a transaction ends abnormally, the AS/400 system can automatically reverse all changes the failed transaction made to the files. This ensures the data is at a known transaction boundary, and incomplete updates are not left in the data base.

The AS/400 system can verify the validity of data for fields defined in the data base. For example, you might define a field as having a valid range of 1000 to 2000. A display file can use this information to verify that the correct information was keyed by the operator. If the operator keys anything outside of the range, the system rejects the data. Your program does not need to check for this range of data values; the system does the checking for you when the data is entered from a display. Only records whose data meets your validity checks are passed to your program.

The AS/400 system allows you to save and restore your data base files, individually and by groups. You can also save and restore all the data on your system.

Joining Data from More Than One File

The AS/400 data base management system allows you to use two or more physically separate files as if they were one file. The data base management system does all the necessary read operations to get the data from the separate files and joins the data to make it appear as if it were coming from a single file. The AS/400 data base management system does the work, not your program.

For example, assume you have an accounts receivable master file and an accounts receivable payment file on your current system. The master file has a record for each of your customers. The master file record format includes the customer number, customer name, and customer address fields. The payment file contains a record for each payment received. Each payment record includes the customer number, payment date, invoice number, and payment amount fields. There is a relationship between the master file data and the payment file data. That is, the record in the master file with customer number 15 may have one or more corresponding payment records in the payment file. Using a traditional file system to get the customer's master record information along with the customer's latest payment record, your program would have to:

1. Read the record in the master file.
2. Read the corresponding record in the payment file.

A data base management system, once you have defined the relationship between the two files, could present the data to the program as if all the data resided in one file. That is, your program would do one read to retrieve the data from the two files.

Selecting and Arranging Data

The AS/400 data base management system can improve productivity by doing some of the functions you would normally have to do in a program. This can reduce your coding effort. For example, the data base management system can select records based on selection values you specify. Your program only sees the records you want it to see. In addition the data base management system can present data in the sequence and groups you specify, without the need to sort or duplicate the data. You can also use the built-in data base system functions to determine, among other things, the standard deviation, variance, square root, and sum of a series of values in the data base. The data base management system can do this work for you, rather than you coding it in a program.

Data Definition

The AS/400 system supplies a common, consistent way of describing data. When you create a data base file, you describe the fields that are contained in that file. All programs that use that file can then use the common definition of the fields in that file. This can help reduce coding errors, makes programs easier to maintain, and helps ensure the consistency of field names and field characteristics.

Summary

Given these characteristics, some of the benefits of using a data base management system include:

- Programs are easier and faster to write and maintain.
- You have better control over the data in your system.
- Data integrity and security are improved.
- Files and applications can easily grow and change.

The Integrated AS/400 Data Base Management System

The AS/400 data base management system supports a rich set of data base functions. However, what sets this system apart from most other systems is that it is **integrated** into the operating system and microcode of the AS/400 system. That is, the data base management system is a fundamental part of the AS/400 system. Any program that wants to store or use data on the AS/400 system uses the integrated data base management system. The integrated data base management system supplies not only a rich set of functions, but also a level of productivity, ease of use, and consistency that many other systems with separate data base management systems cannot achieve. Some of the advantages of using the AS/400 integrated data base management system are:

- No installation required
- Single source of data
- Data available to all programs
- Faster performance
- Improved consistency

No Installation Required

There is no requirement to install or maintain a separate data base management system. When you install the AS/400 system, you install the integrated data base management system.

Single Source of Data

Traditional systems that do not have an integrated data base system usually have multiple ways of storing and retrieving data. The programmer frequently must use one language to access file data (a file interface), and another language to access data base data (a data base interface). In addition, the data stored by the file system often cannot be directly accessed by the data base management system, and the data stored by the data base management system cannot be directly accessed by the file system.

The AS/400 data base serves as the single source of data on the system, and the AS/400 data base manager serves as the single manager for that data. Because there is only one way to store and retrieve data on the system, programs can access *all* the data on the system (provided, of course, the user running the program has the proper authority to use the data). The single data base manager supports several ways to access and manage the data:

- File interface (for example, read and write high-level language statements)
- Data base interface (SQL/400)
- OS/400 IDDU
- OS/400 control language
- System menus

In addition, products such as AS/400 Query, AS/400 Office, AS/400 PC Support, and the AS/400 Application Development Tools access data stored in the data base.

Regardless of the method you choose, you can access all the data on the AS/400 system. For example, you can create a data base using the data base interface, then write a program using the file interface to process the data in that data base. Or, you could create a file using the traditional file interface, then process that data using the data base interface.

In addition, because a file interface is available for the data base management system, you can use many programs originally written for traditional file systems, with little or no change. These traditional programs, when run on the AS/400 system, immediately use the integrated data base management system. Over time, you can change your traditional file programs to take full advantage of the capabilities of the AS/400 data base management system.

Data Available to All Programs

All programs that store or retrieve data on the AS/400 system use its integrated data base management system. Therefore, you can use the data base on the AS/400 system to store data from one program, then use that data in another, completely different application. For example, you can run AS/400 Query to select records from a production data base file and write the results to a new data base file. Then, you can run the AS/400 Business Graphics Utility to produce a chart of the data in that new file. You could then run a high-level language program to change the data in the data base file and, finally, run AS/400 Office to merge the changed data into documents.

Performance

An integrated data base management system normally provides superior performance when compared to the performance of data base management systems added on later. The AS/400 system was designed with data base in mind, so the system is optimized to run its data base management system efficiently.

Consistency

The data base management system uses other AS/400 integrated functions. For example, the data base manager uses the integrated security functions on the AS/400 system. Therefore, once you learn how to use the security function on the system, you will use it to secure data in the data base, just like you use it to secure anything else on the system.

Another example of the benefits of consistency is how the the data base management system communicates with you. The data base system uses the common message handling function on the AS/400 system. Therefore, messages sent by the data base management system to the programmer or operator are in the same, consistent format as messages sent by other functions of the system.

Chapter 2. The AS/400 Data Base Structure

This chapter discusses the basic concepts of the AS/400 data base structure.

Fields and Field Definitions

A **field**, or column, is a named group of values that helps describe an attribute of some thing. For example, the field *Address* is a group of characters that, taken together, describe one attribute of a customer. Other fields that help describe a customer could be: *Custname*, *Age*, *Sex*, and *Amtdue*. A field is the smallest unit of addressable data in the data base. A **field definition** is a description of the field attributes (such as its length or whether the field is numeric or character).

Records and Record Formats

A **record**, or row, is a named group of fields that, taken together, describe some thing. For example, the fields *Custname*, *Custaddr*, *Age*, *Sex*, and *Amtdue* when put together describe a customer. These fields, then, make up a record. A **record format** is simply a description of all the fields in the record and their arrangement in the record. The record format is given a name; some high-level languages may require the use of the record format name in the program.

Data Base Files and File Descriptions

A **file** is a named group of records. For example, the customer master file is the group of all customer master records. A **file description** describes the file including the record format and access path used by the file. (Access paths are described later in this chapter.)

Note: In this guide the terms file and data base file are used interchangeably. In addition, in this guide the term data base refers to all the data base files on the system. However, in SQL/400, the term data base refers to a subset of all the files on the system. For more information about SQL/400 and how it uses the term data base, see the *SQL/400 Programmer's Guide*.

There are two types of data base files: physical files and logical files. Both physical files and logical files may be used to access data in the data base. In fact, in most cases, a programmer does not need to know whether data is accessed through a physical or logical file.

Physical Files

Physical files, or tables, contain the actual data stored on the system. They are similar to traditional files. Each physical file has only one fixed-length record format.

Note: IDDU may be used to make it appear to AS/400 Query, AS/400 PC Support, and the AS/400 Application Development Tools data file utility (DFU) that a physical file contains more than one record format.

A physical file can have a keyed sequence access path to present the data in a sequence other than the order in which the records were added.

Logical Files

Logical files do not actually contain data, but describe how records contained in one or more physical files are to be presented to your program. Some of the things you can do with a logical file are:

- Logically change the attributes of fields from physical files (for example, field length and field order).
- Provide additional logical sequences of records.
- Secure one or more fields in physical files from being read or changed.
- Derive new fields from physical file fields.
- Secure specific physical file records from being read.
- Make two or more files appear as a single file.

There are four categories of logical files:

- A **simple logical file** uses data from one physical file. A simple logical file is the most commonly used category of logical file. It is used to select fields or records from the physical file it is based on. It is also used to arrange the physical file data through a keyed sequence access path. You can read, update, add, and delete records through a simple logical file.
- A **join logical file** combines (in one record format) fields from two or more physical files. A join logical file is read-only; you cannot change, add, or delete records through a join logical file.
- A **multiple format logical file** uses fields from two or more physical files. A multiple format logical file has more than one record format. That is, a record format for each physical file used must be described in the logical file. The records from each of the physical files is presented in a hierarchical fashion. You can read, update, add, or delete data through a multiple format logical file.
- A **view** is created using SQL/400. A view is a logical file without a keyed access path defined for it. One of the most powerful aspects of a view is that it can be created over both physical and logical files. You can read, update, add, or delete data through a view. Views are described and created only through SQL/400 statements; therefore, they are discussed in more detail in the *SQL/400 Programmer's Guide*.

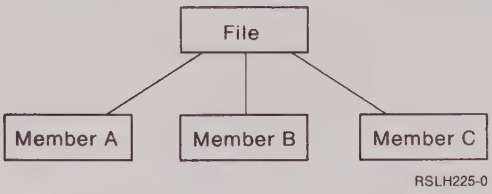
Data and Source Files

Data base files are also categorized as either data files or source files. A **data file** contains the actual data, or a logical view of it, that your programs use. A **source file** contains source specifications the system uses to create an object. For example, to create a program you key the source statements into a source file, and then run the appropriate command to use the source statements to create a program. Source files can be processed in the same way as data files. For more information about using source files, see Chapter 17.

Members

Data records in a data base file are grouped into **members**. All the records in a file can be in one member or they can be grouped in different members.

If you want to process data in a file, the file must have at least one member. The following illustrates the concept of members:



Each member has its associated data and its own access path for that data. The access path specifications and the record formats described in the file are only specified once in the DDS to create the file, and are used for each member's access paths and record formats.

Normally, data base data files have only one member, the one added by default when the file is created. Because, by default, most files are created with one member and most data base commands and operations use it, you do not normally have to use member names when working with files.

Depending on how you use the data in the file, you might want to divide the file into smaller groups of records. If you decide to make smaller groups, you can manage them more easily by assigning a member name to each group. The system can then read only records from that member when you specify the member name.

For example, you define an accounts receivable file. You decide to keep one year's data in that file, but you usually process just one month's data at a time. In this case, you can create a physical file with 12 members, one named for each month. The JANUARY member only contains data for January, the FEBRUARY member only contains data for February, and so on. You can then process each month's data separately, by processing a member at a time. At year end, you can process several (or all) of the members together.

Members are especially useful when storing source statements in a data base file. Members allow you to more easily manage the source. For example, if you had one file that contained the source statements for all your COBOL/400 programs, then you could divide that file into members, with each member containing one program's source statements. So, the PGMA member would contain the source statements for PGMA, the PGMB member would contain the source statements for PGMB, and so on. You could manage each program's source separately through members, without creating multiple files.

Access Paths

When you add data to a physical file, normally new records are physically stored at the end of the file. That is, records are stored in the order that they arrive in the file. However, you may want to read records in an order different from the arrival sequence of the data. For example, you may want to read data in sequence by the *Custname* field.

An **access path** describes the order in which records are to be read. There are two types of access paths: arrival sequence access paths and keyed sequence access paths.

Arrival Sequence Access Path

An **arrival sequence access path** is the order of records as they are stored in the file. Processing files using the arrival sequence access path is similar to processing consecutive, sequential, or direct files on traditional systems.

Keyed Sequence Access Path

A **keyed sequence access path** is the order of records established by the contents of one or more fields in the record. These fields are called key fields. A **key field** is a field whose contents are used to arrange the sequence of the records in the file. Using a keyed sequence access path, the system can present data to a program arranged by the key field of the file. For example, you could retrieve data arranged by the key field *Custname*. Processing files using a keyed sequence access path is similar to processing indexed or keyed files on traditional systems.

A keyed sequence access path is changed whenever records are added to or deleted from a file or whenever a record is changed and the contents of a key field changes. Thus, the access path to the data is automatically maintained by the system.

Chapter 3. Working with the AS/400 Data Base

This chapter describes how you work with data base files on the system. Topics include how you set up data base files, use data base files in a program, and manage data base files. This chapter is only an overview. Detailed information about these topics are covered later in this guide.

Setting Up a Data Base File

There are two steps to setting up a data base file on the system:

1. Describe the data base file to the system.
2. Create the data base file.

Describing the Data Base File

You may choose whether or not to describe the fields in your data base file to the system. If you choose not to describe the fields that make up the record format for the file, then you simply tell the system the length of a record in your physical file (through a parameter on the create file commands). It is then up to you to describe the record format in each of the programs that use that file. A file that must be described by each program that uses it is called a program-described file. This approach is similar to most traditional file systems, but it does not take full advantage of the capabilities of the AS/400 data base management system.

The preferred approach is to describe the file, in detail, to the system one time. This lets you use more of the functions of the data base management system. For example, once the file is described and created, you do not have to code the record format in every program that uses that file. Instead, when you write your program you can request the compiler to automatically extract the field information from the system and include it in your program. A file containing a description of its fields is called an externally described file because the description of the file is maintained as part of the file, external to your programs. Externally described data saves you coding effort, makes it easier to maintain your programs, helps ensure consistent use of field names in your programs, and can provide better documentation.

Externally described and program-described files may also be described in a data dictionary. A **data dictionary** on the AS/400 system is a set of data base files that contain file, record format, and field definitions. A file that is defined in the data dictionary is called a dictionary-described file. IDDU allows you to manage the data dictionary. For more information about the data dictionary and IDDU, see the *IDDU User's Guide*.

A data base file description can contain:

- A description of the record format. The record format describes the fields in the file. Some of the things you can describe for each field include:
  - The field name
  - The size of the field
  - The field data type (for example, packed decimal or character)
  - Text description for each field
  - Validity checks for each field (applies when the field is used in a display file)
  - Editing criteria for each field (applies when the field is used in a display or print file)
- A description of the access path. The access path describes the order in which records are to be presented to programs. Some of the things you can describe for the access path include:
  - If the access path is arrival or keyed sequence
  - The key field(s)
  - If unique or duplicate keys are supported in the file

Creating the Data Base File

After you have described the data base file to the system, you can create the file. When you create a file, you can specify attributes for that file. Some of the attributes you can specify include:

- A text description of the file
- The size of the file
- The amount of time a program should wait if a record in the file is locked by another user or program
- The maximum number of members allowed in the file

You can specify many additional data base file attributes. For more information about file attributes, see "Specifying Data Base File and Member Attributes" on page 4-23.

An Example of Using DDS to Set Up Data Base Files

As a guide to setting up your first physical and logical files using DDS, you might follow these steps:

1. Create a source file (see "Creating a Source File" on page 17-1).
2. Enter the DDS for a physical file into the source file (see "Describing Data Base Files Using DDS" on page 4-6 and "Working with Source Files" on page 17-3).
3. Create the physical file (see "Creating Data Base Files" on page 4-23).
4. Enter the DDS for a logical file into the source file (see "Describing Data Base Files Using DDS" on page 4-6 and "Working with Source Files" on page 17-3).
5. Create the logical file (see "Creating Data Base Files" on page 4-23).

Processing a Data Base File in a Program

To use a data base file in a program, the basic steps are:

1. Determine the run-time attributes of the file.
2. Open the data base file.
3. Process the data in the file.
4. Close the data base file.

Before you open and use a file in a program, you should consider how you plan to use the file. For example, if you attempt to read a record that is locked by someone else, your program normally waits 60 seconds for the record to be released. You may want to wait a shorter or longer period of time. You can control this run-time file attribute, along with other file attributes. It is important to understand these run-time file attributes to take full advantage of the data base management system. A description of the run-time file attributes are discussed in Chapter 8.

A program must open a file before working with the data in that file. An open request connects a file to a program. If you do not specifically request that the file be opened, some languages automatically do it for you when you first refer to the file. For more information about opening a file, see Chapter 9 and your high-level language guide.

After the file is open, the program may proceed to read, update, add, and delete records in the file. For more information about file processing operations, see Chapter 10 and your high-level language guide.

The program indicates that it is finished processing the file by closing the file. A close request disconnects the file from the program. If the program does not specifically request that the file be closed, the language may automatically close the file. For more details about closing data base files, see Chapter 11 and your high-level language guide.

Managing Data Base Files

The AS/400 system has a number of commands to help you manage your data base system. These include commands to:

- Create and change data base files and members
- Display data base cross-reference information
- Start and stop data base recovery functions

You can also use the Work with Files (WRKF) command and menus to help manage your data base files. The WRKF command has menus that allow you to:

- Copy data from a file
- Delete a file
- Display the records in a file member
- Display the description of the file
- Save the file
- Restore the file
- Change certain file attributes

Member Operations

Because data is kept in file members, the system supplies you with a number of functions to manage files and members:

- Add members to a data base file
- Rename members
- Remove members from a data base file
- Change processing attributes of data base file members
- Initialize data in physical file members
- Clear data in physical file members
- Reorganize data in physical file members
- Display data in physical file members

These commands are discussed in more detail in Chapter 13.

Data Base Cross-Reference Information

The AS/400 system contains information to assist you in cross-referencing file and program use. This information can be very important to programmers in determining such things as:

- What files are used in specified programs
- What files depend on other files for data or access paths
- The attributes of specified files
- The fields defined in a specified file or record format

The programmer can use this information to help determine the effect that a change to a field or file might have on other programs in the system.

Some of the system commands can write the results of the command to a data base file. Then, you can run a program to analyze the results.

These commands and functions are discussed in more detail in Chapter 15. If a file is described using IDDU, it can provide cross-reference information. For more information about the cross-reference information available through IDDU, see the *IDDU User's Guide*.

Data Base Recovery

It is always a good data processing practice to be prepared to recover your data should the system fail for any reason. The AS/400 system supplies a number of integrated functions to help you recover your data base files quickly. These functions include:

- A journal to record changes to your data base files that can be used to help recover your data and access paths.
- A function, known as commitment control, to help ensure that, should a job end abnormally, any changes made to the data base that are incomplete are rolled-back to a known, clean state (known as a transaction boundary).
- A comprehensive set of commands to save and restore your data.

For more detail about the AS/400 data base recovery functions, see Chapter 16 of this guide and the *Backup and Recovery Guide*.

Remote File Access

Programs on the AS/400 system can access files on remote systems through OS/400 DDM. **Distributed data management** (DDM) allows remote files to appear to the programmer and end user as local data base files. Programs using DDM do *not* need special or additional programming statements. The fact that the program is accessing data on a remote system, as opposed to the local system, is transparent to the program.

DDM files have the following characteristics:

- The AS/400 system supports read, add, update, and delete operations to remote data base files. Programs and commands using DDM files run as though the remote data base files were local data base files.
- The DDM file, in effect, represents a remote data base file. The DDM file references the communications path and the remote file, and links the remote system to the local system when the file is opened.

For more information about DDM, see the *DDM User's Guide*.

Part 2. Setting Up Data Base Files

Chapter 4. General Considerations

This chapter discusses things to consider when you set up any AS/400 data base file. Later chapters will discuss unique considerations for setting up physical and logical files.

Describing Data Base Files

Records in data base files can be described in two ways:

- Field level description. The fields in the record are described to the system. Some of the things you can describe for each field include: name, length, data type, validity checks, and text description. Data base files that are created with field level descriptions are referred to as externally described files.
- Record level description. Only the length of the record in the file is described to the system. The system does *not* know about fields in the file. These data base files are referred to as program-described files.

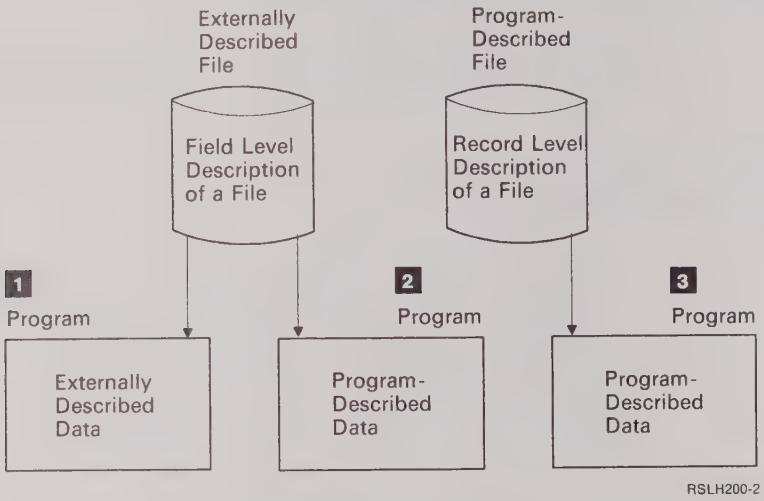
Regardless of whether a file is described to the field or record level, you must describe and create the file before you can compile a program that uses that file. That is, the file must exist on the system before you use it.

Programs can use either externally described or program-described files. However, if you choose to describe a file to the field level, the system can do more for you. For example, when you compile your programs, the system can extract information from an externally described file and automatically include field information in your programs. You do not have to code the field information in each program.

Programs can use file descriptions in two ways:

- The program uses the field-level description that is part of the file. Because the field descriptions are external to the program itself, the term, externally described data, is used.
- The fields are described in the program that processes the file; therefore, the data is called program-described data. Files that are only described to the record level must be described in the program using the file.

The following figure shows the typical relationships between files and programs on the AS/400 system:



1. The program uses the field level description of a file that is defined to the system. At compilation time, the language compiler copies the external description of the file into the program.
2. The program uses a file that is described to the field level to the system, but it does not use the actual field descriptions. At compilation time, the language compiler does not copy the external description of the file into the program. The fields in the file are described in the program. In this case, the field attributes (for example, field length) used in the program must be the same as the field attributes in the external description.
3. The program uses a file that is described only to the record level to the system. The fields in the file must be described in the program.

Externally described files may also be described in a program. You might want to use this method for compatibility with previous systems. For example, you want to run programs on the AS/400 system that originally came from a traditional file system. Those programs use program-described data, and the file itself is only described to the record level. At a later time, you describe the file to the field level (externally described file) in order to use more of the data base functions available on the system. Your old programs, containing program-described data, can continue to use the externally described file while new programs use the field-level descriptions that are part of the file. Over time, you can change one or more of your old programs to use the field level descriptions.

Dictionary-Described Data

A program-described file can be dictionary-described. You can describe the record format information using IDDU. Even though the file is program-described, AS/400 Query, AS/400 PC Support, and DFU will use the record format description stored in the data dictionary.

An externally described file can also be dictionary-described. You can use IDDU to describe a file, then create the file using IDDU. The file created is an externally described file. You can also move into the data dictionary the file description stored in an externally described file. The system always ensures that the definitions in the

data dictionary and the description stored in the externally described file are identical.

For more information about the data dictionary and IDDU, see the *IDDU User's Guide*.

Methods of Describing Data to the System

If you want to describe a file just to the record level, you can use the record length (RCDLEN) parameter on the Create Physical File (CRTPF) and Create Source Physical File (CRTSRCPF) commands.

If you want to describe your file to the field level, there are several methods of describing data to the data base system: IDDU, SQL/400 commands, or DDS.

Note: Because DDS has the most options for defining data for the programmer, this guide will focus on describing data base files using DDS.

OS/400 IDDU

Physical files can be described using IDDU. You might use IDDU because it is a menu-driven, interactive method of describing data. You also might be familiar with describing data using IDDU on a System/36. In addition, IDDU allows you to describe multiple-format physical files for use with Query, AS/400 PC Support, and DFU.

When you use IDDU to describe your files, the file definition becomes part of the OS/400 data dictionary. For more information about the data dictionary and IDDU, see the *IDDU User's Guide*.

Structured Query Language/400

The Structured Query Language/400 can be used to describe an AS/400 data base file. SQL/400 supports statements to describe the fields in the data base file, and to create the file.

You might use SQL/400 because it is the IBM Systems Application Architecture data base language for defining and processing data base data.

For more information about SQL/400, see the *SQL/400 Programmer's Guide*.

OS/400 DDS

Externally described data files can be described using DDS. Using DDS, you provide descriptions of the field, record, and file level information.

You might use DDS because it provides the most options for the programmer to describe data in the data base. For example, only with DDS can you describe key fields in logical files.

The *DDS Coding Form*, shown in Figure 4-1 on page 4-4, provides a common format for describing data externally.

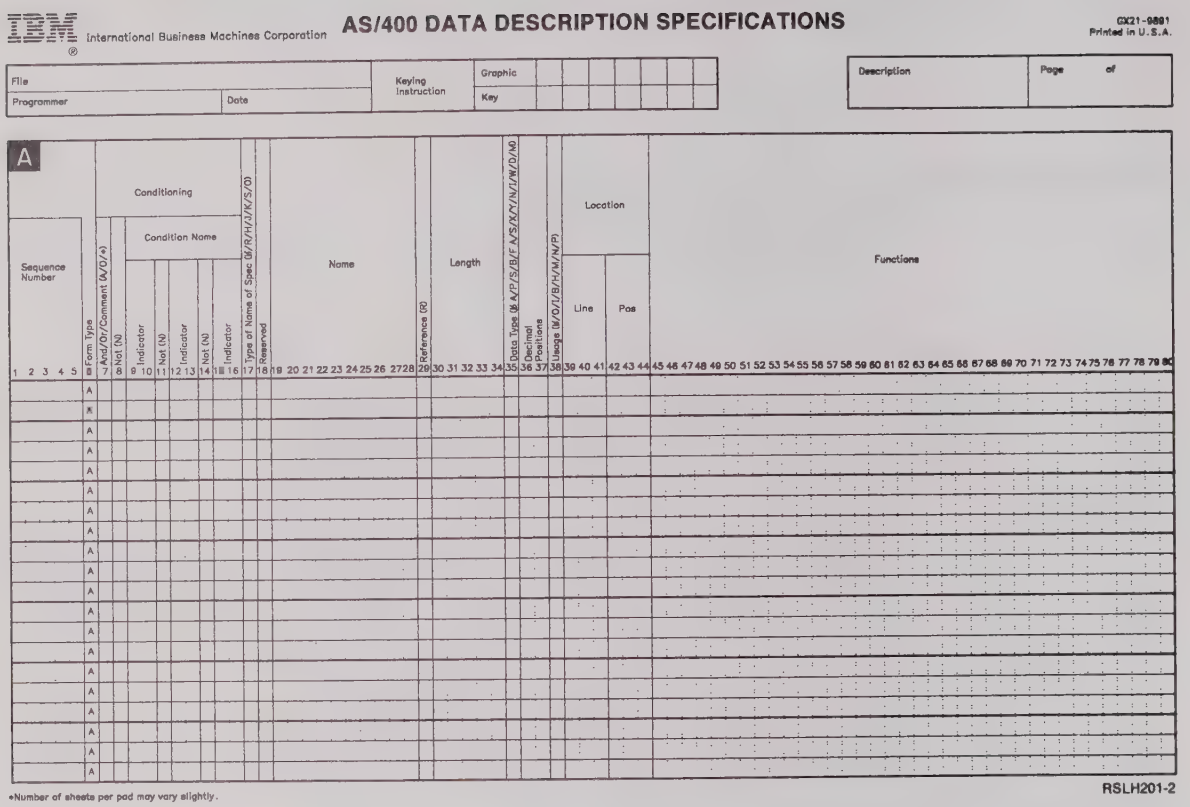


Figure 4-1. Sample DDS Form

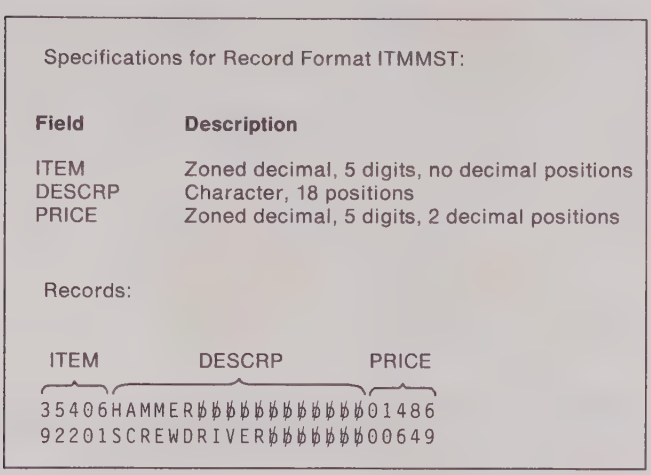
The *DDS Reference* contains a detailed description of how to use DDS for describing physical and logical files.

Describing a Data Base File to the System

When you describe a data base file to the system, you describe the two major parts of that file: the record format and the access path.

Describing the Record Format: The record format describes the order of the fields in each record. The record format also describes each field in detail including: length, data type (for example, packed decimal or character), validity checks, text description, and other information.

The following example shows the relationship between the record format and the records in a physical file:



RSLH203-1

A physical file can have only one record format. The record format in a physical file describes the way the data is actually stored. Physical files do not support variable length fields; therefore, all records in a physical file are fixed length and have the same field descriptions.

A logical file contains no data. Logical files are used to arrange data from one or more physical files into different formats and sequences. For example, a logical file could change the order of the fields in the physical file, or present to the program only some of the fields stored in the physical file.

A logical file record format can change the length and data type of fields stored in physical files. The system does the necessary conversion between the physical file field description and the logical file field description. For example, a physical file could describe FLDA as a packed decimal field of 5 digits and a logical file using FLDA might redefine it as a zoned decimal field of 7 digits. In this case, when your program used the logical file to read a record, the system would automatically convert (unpack) FLDA to zoned decimal format.

Describing the Access Path: An access path describes the order in which records are to be retrieved. When you describe an access path, you describe whether it will be a keyed sequence or arrival sequence access path. Access paths are discussed in more detail in "Describing the Access Path for the File" on page 4-16.

Naming Conventions

The file name, record format name, and field name can be as long as 10 characters and must follow all system naming conventions, but you should keep in mind that some high-level languages have more restrictive naming conventions than the system does. For example, RPG/400 allows only 6-character names, while the system allows 10-character names. In some cases, you can temporarily change (rename) the system name to one that meets the high-level language restrictions. For more information about renaming data base fields in programs, see your high-level language guide.

In addition, names must be unique as follows:

- Field names must be unique in a record format.
- Record format names and member names must be unique in a file.
- File names must be unique in a library.

Describing Data Base Files Using DDS

When you describe a data base file using DDS, you can describe information at the file, record format, field, key, and select/omit levels:

- File level DDS give the system information about the entire file. For example, you can specify whether all the key field values in the file must be unique.
- Record format level DDS give the system information about a specific record format in the file. For example, when you describe a logical file record format, you can specify the physical file that it is based on.
- Field level DDS give the system information about individual fields in the record format. For example, you can specify the name and attributes of each field.
- Key field level DDS give the system information about the key fields for the file. For example, you can specify which fields in the record format are to be used as key fields.
- Select/omit field level DDS give the system information about which records are to be returned to the program when processing the file. Select/omit specifications apply to logical files only.

Example of Describing a Physical File Using DDS

The DDS for a physical file must be in the following order (Figure 4-2):

1. File level keywords (optional). The UNIQUE keyword is used to indicate that the value of the key field in each record in the file must be unique. Duplicate key values are not allowed in this file.
2. Record format level keywords. The record format name is specified, along with an optional text description.
3. Field level keywords. The field names and field lengths are specified, along with an optional text description for each field.
4. Key field level keywords (optional). The field names used as key fields are specified.

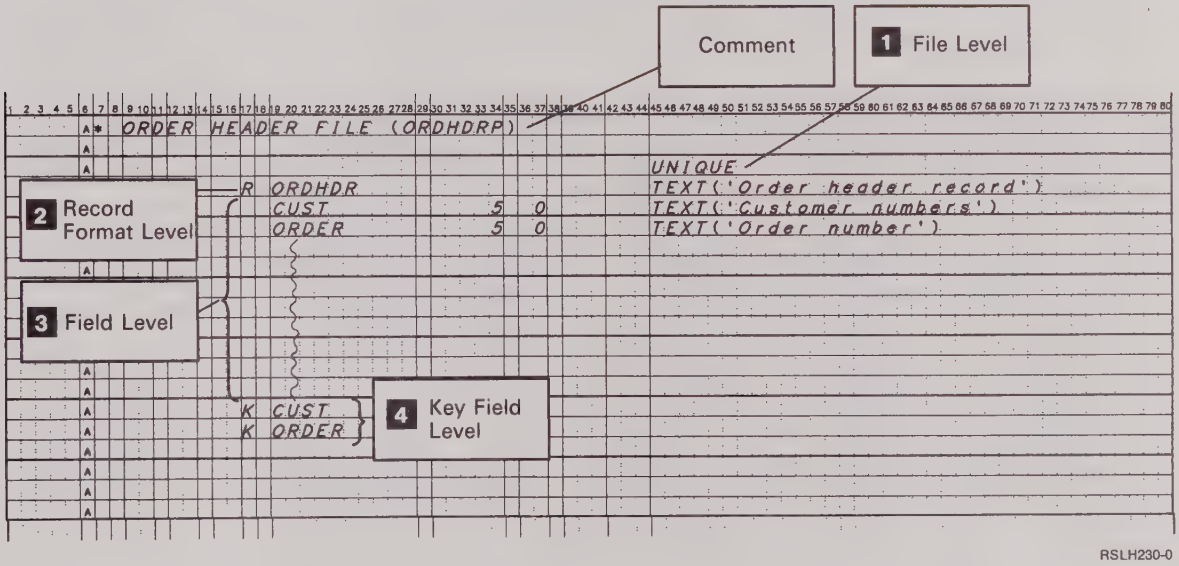


Figure 4-2. DDS for a Physical File (ORDHDRP)

The following example shows a physical file ORDHDRP (an order header file), which has an arrival sequence access path, and the DDS necessary to describe that file.

Record Format (ORDHDR):

Customer Number (CUST)	Order Number (ORDER)	Order Date (ORDATE)	Purchase Order Number (CUSORD)	Shipping Instructions (SHPVIA)	Order Status (ORDSTS)	...	State (STATE)
Packed Decimal Length 5 No Decimals	Packed Decimal Length 5 No Decimals	Packed Decimal Length 6 No Decimals	Packed Decimal Length 15 No Decimals	Character Length 15	Character Length 1		Character Length 1

RSLH231-5

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
     A* ORDER HEADER FILE (ORDHDRP)
     A            R ORDHDR                       TEXT('Order header record')
     A              CUST                5  0     TEXT('Customer Number')
     A              ORDER               5  0     TEXT('Order Number')
     A              ORDATE              6  0     TEXT('Order Date')
     A              CUSORD             15  0     TEXT('Customer Purchase Order No.')
     A              SHPVIA             15        TEXT('Shipping Instructions')
     A              ORDSTS              1        TEXT('Order Status')
     A              OPRNME             10        TEXT('Operator Name')
     A              ORDAMT              9  2     TEXT('Order Amount')
     A              CUTYPE              1        TEXT('Customer Type')
     A              INVNBR              5  0     TEXT('Invoice Number')
     A              PRTDAT              6  0     TEXT('Printed Date')
     A              SEQNBR              5  0     TEXT('Sequence Number')
     A              OPNSTS              1        TEXT('Open Status')
     A              LINES               3  0     TEXT('Order Lines')
     A              ACTMTH              2  0     TEXT('Accounting Month')
     A              ACTYR               2  0     TEXT('Accounting Year')
     A              STATE               2        TEXT('State')
     A
     A
```

RSLH232-0

The R in position 17 indicates that a record format is being defined. The record format name ORDHDR is specified in positions 19 through 28.

You make no entry in position 17 when you are describing a field; a blank in position 17 along with a name in positions 19 through 28 indicates a field name.

The data type is specified in position 35. The valid data types are:

Entry	Meaning
A	Character
P	Packed decimal
S	Zoned decimal
B	Binary
F	Floating point

Notes:

1. For double-byte character set (DBCS) data types, see Appendix B.
2. The AS/400 system performs arithmetic operations more efficiently for packed decimal than for zoned decimal.
3. Some high-level languages do not support floating-point data.
4. Some special considerations that apply when you are using floating-point fields are:
   - The precision associated with a floating-point field is a function of the number of bits (single or double precision) and the internal representation of the floating-point value. This translates into the number of decimal digits supported in the significand and the maximum values that can be represented in the floating-point field.
   - When a floating-point field is defined with fewer digits than supported by the precision specified, that length is only a presentation length and has no effect on the precision used for internal calculations.
   - Although floating-point numbers are accurate to 7 (single) or 15 (double) decimal digits of precision, you can specify up to 9 or 17 digits. You can use the extra digits to uniquely establish the internal bit pattern in the internal floating-point format so identical results are obtained when a floating-point number in internal format is converted to decimal and back again to internal format.

If the data type (position 35) is not specified, the decimal positions entry is used to determine the data type. If the decimal positions (positions 36 through 37) are blank, the data type is assumed to be character (A); if these positions contain a number 0 through 31, the data type is assumed to be packed decimal (P).

The length of the field is specified in positions 30 through 34, and the number of decimal positions (for numeric fields) is specified in positions 36 and 37. If a packed or zoned decimal field is to be used in a high-level language program, the field length must be limited to the length allowed by the high-level language you are using. The length is not the length of the field in storage but the number of digits or characters specified externally from storage. For example, a 5-digit packed decimal field has a length of 5 specified in DDS, but it uses only 3 bytes of storage. For more information on the length of fields in storage, see the *DDS Reference*.

Example of Describing a Logical File Using DDS

The DDS for a logical file must be in the following order (Figure 4-3):

1. File level keywords (optional). In this example, the UNIQUE keyword indicates that for this file the key value for each record must be unique; no duplicate key values are allowed.

For each record format:

2. Record level keywords. In this example, the record format name, the associated physical file, and an optional text description are specified.
3. Field level keywords (optional). In this example, each field name used in the record format is specified.
4. Key field level keywords (optional). In this example, the *Order* field is used as a key field.
5. Select/omit field level keywords (optional). In this example, all records whose *Opnsts* field contains a value of N are omitted from the file's access path. That is, programs reading records from this file will never see a record whose *Opnsts* field contains an N value.

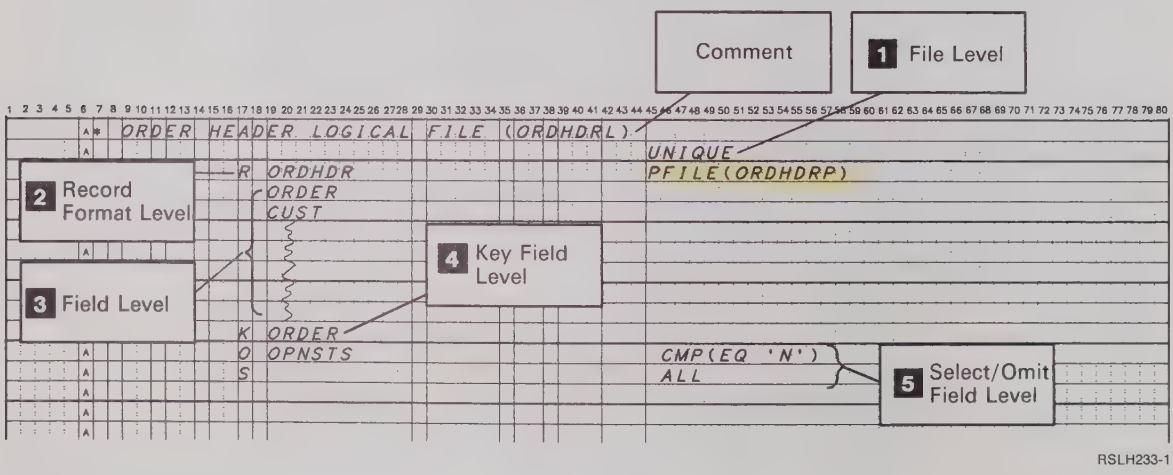


Figure 4-3. DDS for a Simple Logical File (ORDHDRL)

A logical file must be created after all physical files on which it is based are created. The PFILE keyword in the previous example is used to specify the physical file or files on which the logical file is based.

Record formats in a logical file can be:

- A new record format based on fields from a physical file
- The same record format as in a previously described physical or logical file (see "Sharing Existing Record Format Descriptions" on page 4-14)

Fields in the logical file record format must either appear in the record format of at least one of the physical files or be derived from the fields of the physical files on which the logical file is based.

For more information about describing logical files, see Chapter 6.

Additional Field Definition Functions

You can describe additional information about the fields in the physical and logical file record formats with function keywords (positions 45 through 80 on the *DDS Coding Form*). Some of the things you can specify include:

- Validity checking keywords to verify that the field data meets your criteria. For example, you can describe a field to have a valid range of 500 to 900. (This checking is done only when data is typed on a keyboard to the display.)
- Editing keywords to control how a field should be displayed or printed. For example, you can use the EDTCDE(Y) keyword to specify that a date field is to appear as MM/DD/YY. The EDTCDE and EDTWRD keywords can be used to control editing. (This editing is done only when used in a display or printer file.)
- Documentation, heading, and name control keywords to control the description and name of a field. For example, you can use the TEXT keyword to document a description of each field. This text description is included in your compiler listing to better document the files used in your program. The TEXT and COLHDG keywords control text and column-heading definitions. The ALIAS keyword can be used to provide a more descriptive name for a field. The alias, or alternative name, is used in a program (if the high-level language supports alias names).

For more information about how to specify keyword functions, see the *DDS Reference*.

Using Existing Field Descriptions and Field Reference Files

If a field was already described in an existing file, and you want to use that field description in a new file you are setting up, you can request the system to copy that description into your new file description. The DDS keywords REF and REFFLD allow you to refer to a field description in an existing file. This helps reduce the effort of coding DDS statements. It also helps ensure that the field attributes are used consistently in all files that use the field.

In addition, you can create a physical file for the sole purpose of using its field descriptions. That is, the file does not contain data; it is used only as a reference for the field descriptions for other files. This type of file is known as a field reference file. A **field reference file** is a physical file containing no data, just field descriptions.

You can use a field reference file to simplify record format descriptions and to ensure field descriptions are used consistently. You can define all the fields you need for an application or any group of files in a field reference file. You can create a field reference file using DDS and the Create Physical File (CRTPF) command.

After the field reference file is created, you can build physical file record formats from this file without describing the characteristics of each field in each file. When you build physical files, all you need to do is refer to the field reference file (using the REF and REFFLD keywords) and specify any changes. Any changes to the field descriptions and keywords specified in your new file override the descriptions in the field reference file.

In the following example, a field reference file named DSTREFP is created for distribution applications. Figure 4-4 on page 4-12 shows the DDS needed to describe DSTREFP.

```
A*  FIELD REFERENCE FILE (DSTREFP)
A                R DSTREF                    TEXT('Field reference file')
A
A*  FIELDS DEFINED BY CUSTOMER MASTER RECORD (CUSMST)
A                CUST            5  0       TEXT('Customer numbers')
A                                           COLHDG('CUSTOMER' 'NUMBER')
A                NAME           20          TEXT('Customer name')
A                ADDR         R             REFFLD(NAME *SRC)
A                                           TEXT('Customer address')
A                CITY         R             REFFLD(NAME *SRC)
A                                           TEXT('Customer city')
A                STATE           2          TEXT('State abbreviation')
A                                           CHECK(MF)
A                CRECHK          1          TEXT('Credit check')
A                                           VALUES('Y' 'N')
A                SEARCH          6N 0       TEXT('Customer name search')
A                                           COLHDG('SEARCH CODE')
A                ZIP             5  0       TEXT('Zip code')
A                                           CHECK(MF)
A                CUTYPE          1S         COLHDG('CUSTOMER' 'TYPE')
A                                           RANGE(1 5)
A
A*  FIELDS DEFINED BY ITEM MASTER RECORD (ITMAST)
A                ITEM            5          TEXT('Item number')
A                                           COLHDG('ITEM ' 'NUMBER')
A                                           CHECK(M10)
A                DESCRP         18          TEXT('Item description')
A                PRICE           5  2       TEXT('Price per unit')
A                                           EDTCDE(J)
A                                           CMP(GT 0)
A                                           COLHDG('PRICE')
A                ONHAND          5  0       TEXT('On hand quantity')
A                                           EDTCDE(Z)
A                                           CMP(GE 0)
A                                           COLHDG('ON HAND')
A                WHSLOC          3          TEXT('Warehouse location')
A                                           CHECK(MF)
A                                           COLHDG('BIN NO')
A                ALLOC        R             REFFLD(ONHAND *SRC)
A                                           TEXT('Allocated quantity')
A                                           CMP(GE 0)
A                                           COLHDG('ALLOCATED')
A
A*  FIELDS DEFINED BY ORDER HEADER RECORD (ORDHDR)
A                ORDER           5  0       TEXT('Order number')
A                                           COLHDG('ORDER' 'NUMBER')
A                ORDATE          6  0       TEXT('Order date')
A                                           EDTCDE(Y)
A                                           COLHDG('DATE' 'ORDERED')
A                CUSORD         15          TEXT('Customer purchase order numbe+
A                                           r')
A                                           COLHDG('P.O.' 'NUMBER')
A                SHPVIA         15          TEXT('Shipping instructions')
A                ORDSTS          1          TEXT('Order status code')
A                                           COLHDG('ORDER' 'STATUS')
A                OPRNME         10          TEXT('Operator name')
A                                           COLHDG('OPERATOR NAME')
A                ORDAMT          9  2       TEXT('Total order value')
A                                           COLHDG('ORDER' 'AMOUNT')
A
```

RSLH226-0

Figure 4-4 (Part 1 of 2). DDS for a Field Reference File (DSTREFP)

```
     A                INVNBR         5  0          TEXT('Invoice number')
     A                                             COLHDG('INVOICE' 'NUMBER')
     A                PRTDAT         6  0          EDTCDE(Y)
     A                                             COLHDG('PRINTED' 'DATE')
     A                SEQNBR         5  0          TEXT('Sequence number')
     A                                             COLHDG('SEQ' 'NUMBER')
     A                OPNSTS         1            TEXT('Open status')
     A                                             COLHDG('OPEN' 'STATUS')
     A                LINES          3  0          TEXT('Total lines on invoice')
     A                                             COLHDG('TOTAL' 'LINES')
     A                ACTMTH         2  0          TEXT('Accounting month')
     A                                             COLHDG('ACCT' 'MONTH')
     A                ACTYR          2  0          TEXT('Accounting year')
     A                                             COLHDG('ACCT' 'YEAR')
     A
     A*  FIELDS DEFINED BY ORDER DETAIL/LINE ITEM RECORD (ORDDTL)
     A                LINE           3  0          TEXT('Line number of this ordered i+
     A                                             tem')
     A                                             COLHDG('LINE NO')
     A                QTYORD         3  0          TEXT('Quantity ordered')
     A                                             COLHDG('QTY' 'ORDERED')
     A                                             CMP(GE 0)
     A                EXTENS         6  2          TEXT('Extension of QTYORD x PRICE')
     A                                             EDTCDE(J)
     A                                             COLHDG('EXTENSION')
     A
     A*  FIELDS DEFINED BY ACCOUNTS RECEIVABLE
     A                ARBAL          8  2          TEXT('A/R balance due')
     A                                             EDTCDE(J)
     A
     A*  WORK AREAS AND OTHER FIELDS THAT OCCUR IN MULTIPLE PROGRAMS
     A                STATUS        12            TEXT('Status description')
     A
     A
```

RSLH227-0

Figure 4-4 (Part 2 of 2). DDS for a Field Reference File (DSTREFP)

To then create a field reference file, use the Create Physical File (CRTPF) command. Assume that the DDS in Figure 4-4 on page 4-12 was entered into a source file FRSOURCE; the member name is DSTREFP. The parameter MBR(*NONE) tells the system not to add a member to the file (because the field reference file will never contain data and therefore does not need a member).

```
CRTPF FILE(DSTPRODLB/DSTREFP)
      SRCFILE(QGPL/FRSOURCE) MBR(*NONE)
      TEXT('Distribution field reference file')
```

To describe the physical file ORDHDRP by referring to DSTREFP, use the following DDS (Figure 4-5):

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
     A*  ORDER HEADER FILE (ORDHDRP) -  PHYSICAL FILE RECORD DEFINITION
     A                                  REF(DSTREFP)
     A            R ORDHDR              TEXT('Order header record')
     A              CUST           R
     A              ORDER          R
     A              ORDATE         R
     A              CUSORD         R
     A              SHPVIA         R
     A              ORDSTS         R
     A              OPRNME         R
     A              ORDAMT         R
     A              CUTYPE         R
     A              INVNBR         R
     A              PRTDAT         R
     A              SEQNBR         R
     A              OPNSTS         R
     A              LINES          R
     A              ACTMTH         R
     A              ACTYR          R
     A              STATE          R
                                                                    RSLH228-1
```

Figure 4-5. DDS for a Physical File (ORDHDRP) Built From a Field Reference File

The REF keyword (positions 45 through 80) with DSTREF (the field reference file name) specified indicates the file from which field descriptions are to be used. The R in position 29 of each field indicates that the field description is to be taken from the reference file.

When you create the ORDHDRP file, the system will use the DSTREF file to determine the attributes of the fields included in the ORDHDR record format. To create the ORDHDRP file, use the Create Physical File (CRTPF) command. Assume that the DDS in Figure 4-5 was entered into a source file QDDSSRC; the member name is ORDHDRP.

```
CRTPF FILE(DSTPRODLB/ORDHDRP)
      TEXT('Order Header physical file')
```

Note: The files used in some of the examples in this guide refer to this field reference file.

Using a Data Dictionary for Field Reference

You can use a data dictionary as an alternative to using a field reference file. IDDU allows you to define fields in a data dictionary. For more information about data dictionaries and IDDU, see the *IDDU User's Guide*.

Sharing Existing Record Format Descriptions

A record format can be described once in either a physical or logical file (except a join logical file) and can be used by many files. When you describe a new file, you can specify that the record format of an existing file is to be used by the new file. This can help reduce the number of DDS statements that you would normally code to describe a record format in a new file and can save auxiliary storage space.

The file originally describing the record format can be deleted without affecting the files sharing the record format. After the last file using the record format is deleted, the system automatically deletes the record format description.

The following shows the DDS for two files. The first file describes a record format, and the second shares the record format of the first:

```
A                R RECORD1                   PFILE(CUSMSTP)
A                  CUST
A                  NAME
A                  ADDR
A                  SEARCH
A                K CUST
```

RSLH208-0

Figure 4-6. DDS for a Logical File (CUSMSTL)

```
A                R RECORD1                   PFILE(CUSMSTP)
A                                            FORMAT(CUSMSTL)
A                K NAME
```

RSLH209-0

Figure 4-7. DDS for a Logical File (CUSTMSTL1) Sharing a Record Format

The example shown in Figure 4-6 shows file CUSMSTL, in which the fields *Cust*, *Name*, *Addr*, and *Search* make up the record format. The *Cust* field is specified as a key field.

The DDS in Figure 4-7 shows file CUSTMSTL1, in which the FORMAT keyword names CUSMSTL to supply the record format. The record format name must be RECORD1, the same as the record format name shown in Figure 4-6. Because the files are sharing the same format, both files have fields *Cust*, *Name*, *Addr*, and *Search* in the record format. In file CUSMSTL1, a different key field, *Name* is specified.

The following restrictions apply to shared record formats:

- A physical file cannot share the format of a logical file.
- A join logical file cannot share the format of another file, and another file cannot share the format of a join logical file.
- A view cannot share the format of another file, and another file cannot share the format of a view.

If the original record format is changed by deleting all related files and creating the original file and all the related files again, it is changed for all files that share it. If only the file with the original format is deleted and re-created with a new record format, all files previously sharing that file's format continue to use the original format.

If a logical file is defined but no field descriptions are specified and the FORMAT keyword is not specified, the record format of the first physical file (specified first on the PFILE keyword for the logical file) is automatically shared. The record format name specified in the logical file must be the same as the record format named specified in the physical file.

To find out if a file shares a format with another file, use the RCDFMT parameter on the Display Data Base Relations (DSPDBR) command.

Describing the Access Path for the File

An access path describes the order in which records are to be retrieved. Records in a physical or logical file can be retrieved using an arrival sequence access path or a keyed sequence access path. For logical files, you can also select and omit records based on the value of one or more fields in each record.

Arrival Sequence Access Path

The arrival sequence access path is based on the order in which the records arrive and are stored in the file. For reading or updating, records can be accessed:

- Sequentially, where each record is taken from the next sequential physical position in the file.
- Directly by relative record number, where the record is identified by its position from the start of the file.

An externally described file has an arrival sequence access path when no key fields are specified for the file. A program-described file always has an arrival sequence access path.

An arrival sequence access path is valid only for the following:

- Physical files.
- Logical files in which each member of the logical file is based on only one physical file member.
- Join logical files.
- Views.

Notes:

1. Arrival sequence is the only processing method that allows a program to use the storage space previously occupied by a deleted record by placing another record in that storage space. For more information about processing deleted records, see "Deleting Data Base Records" on page 10-9.
2. Through your high-level language, the Display Physical File Member (DSPPFM) command, and the Copy File (CPYF) command, you may be able to process a keyed sequence file in arrival sequence. You can use this function for a physical file, a simple logical file based on one physical file member, or a join logical file.
3. Through your high-level language, you may be able to process a keyed sequence file directly by relative record number. You can use this function for a physical file, a simple logical file based on one physical file member, or a join logical file.
4. An arrival sequence access path does not take up any additional storage and is always saved or restored with the file. (Because the arrival sequence access path is nothing more than the physical order of the data as it was stored, when you save the data you save the arrival sequence access path.)

Keyed Sequence Access Path

A keyed sequence access path is based on the contents of the key fields as defined in DDS. This type of access path is updated whenever records are added or deleted, or when records are updated and the contents of a key field is changed. The keyed sequence access path is valid for both physical and logical files. The sequence of the records in the file is defined in DDS when the file is created and is maintained automatically by the system.

Key fields defined as character fields are arranged based on the sequence defined for EBCDIC characters. Key fields defined as numeric fields are arranged based on their algebraic values, unless the DDS UNSIGNED (unsigned value) or ABSVAL (absolute

value) keywords are specified for the field. Key fields defined as DBCS are allowed, but are arranged only as single bytes based on their bit representation.

Arranging Key Fields in Ascending or Descending Sequence: Key fields can be arranged in either ascending or descending sequence. Consider the following records:

Record	Empnbr	Clsnbr	Clsnam	Cpdate
1	56218	412	Welding I	032188
2	41322	412	Welding I	011388
3	64002	412	Welding I	011388
4	23318	412	Welding I	032188
5	41321	412	Welding I	051888
6	62213	412	Welding I	032188

If the *Empnbr* field is the key field, there are two possibilities for organizing these records:

- In ascending sequence, the order of the records in the access path is:

Record	Empnbr	Clsnbr	Clsnam	Cpdate
4	23318	412	Welding I	032188
5	41321	412	Welding I	051888
2	41322	412	Welding I	011388
1	56218	412	Welding I	032188
6	62213	412	Welding I	032188
3	64002	412	Welding I	011388

- In descending sequence, the order of the records in the access path is:

Record	Empnbr	Clsnbr	Clsnam	Cpdate
3	64002	412	Welding I	011388
6	62213	412	Welding I	032188
1	56218	412	Welding I	032188
2	41322	412	Welding I	011388
5	41321	412	Welding I	051888
4	23318	412	Welding I	032188

When you describe a key field, the default is ascending sequence. However, you can use the DESCEND DDS keyword to specify that you want to arrange a key field in descending sequence. For more information about how to specify this keyword, see the *DDS Reference*.

Using More Than One Key Field: You can use more than one key field to arrange the records in a file. The key fields do not have to use the same sequence. For example, when you use two key fields, one field can use ascending sequence while the other can use descending sequence. Consider the following records:

Record	Order	Ordate	Line	Item	Qtyord	Extens
1	52218	063088	01	88682	425	031875
2	41834	062888	03	42111	30	020550
3	41834	062888	02	61132	4	021700
4	52218	063088	02	40001	62	021700
5	41834	062888	01	00623	50	025000

If the access path uses the *Order* field, then the *Line* field as the key fields, both in ascending sequence, the order of the records in the access path is:

Record	Order	Ordate	Line	Item	Qtyord	Extens
5	41834	062888	01	00623	50	025000
3	41834	062888	02	61132	4	021700
2	41834	062888	03	42111	30	020550
1	52218	063088	01	88682	425	031875
4	52218	063088	02	40001	62	021700

If the access path uses the key field *Order* in ascending sequence, then the *Line* field in descending sequence, the order of the records in the access path is:

Record	Order	Ordate	Line	Item	Qtyord	Extens
2	41834	062888	03	42111	30	020550
3	41834	062888	02	61132	4	021700
5	41834	062888	01	00623	50	025000
4	52218	063088	02	40001	62	021700
1	52218	063088	01	88682	425	031875

When a record has key fields whose contents are the same as the key field in another record in the same file, then the file is said to have records with duplicate key values. However, the duplication must occur for *all* key fields for a record if they are to be called duplicate key values. For example, if a record format has two key fields *Order* and *Ordate*, duplicate key values occur when the contents of both the *Order* and *Ordate* fields are the same in two or more records. There are duplicate key values for the following records:

First Key Field Order	Second Key Field Ordate	Line	Item	Qtyord	Extens
41834	062888	03	42111	30	020550
41834	062888	02	61132	04	021700
41834	062888	01	00623	50	025000

Using the *Line* field as a third key field defines the file so that there are no duplicate keys:

First Key Field Order	Second Key Field Ordate	Third Key Field Line	Item	Qtyord	Extens
41834	062888	03	42111	30	020550
41834	062888	02	61132	04	021700
41834	062888	01	00623	50	025000

A logical file that has more than one record format can have records with duplicate key values, even though the record formats are based on different physical files. That is, even though the key values come from different record formats, they are considered duplicate key values.

Preventing Duplicate Key Values: The AS/400 data base management system allows records with duplicate key values in your files. However, you may want to prevent duplicate key values in some of your files. For example, you may create a file where the key field is defined as the customer number field. In this case, you want the system to ensure that each record in the file has a unique customer number.

You can prevent duplicate key values in your files by specifying the UNIQUE keyword in DDS. With the UNIQUE keyword specified, a record cannot be entered or copied into a file if its key value is the same as the key value of a record already existing in the file.

If records with duplicate key values already exist in a physical file, the associated logical file cannot have the UNIQUE keyword specified. If you try to create a logical file with the UNIQUE keyword specified, and the associated physical file contains duplicate key values, the logical file is not created. The system sends you a message stating this and sends you messages (as many as 20) indicating which records contain duplicate key values.

When the UNIQUE keyword is specified for a file, any record added to the file cannot have a key value that duplicates the key value of an existing record in the file, regardless of the file used to add the new record. For example, two logical files LF1 and LF2 are based on the physical file PF1. The UNIQUE keyword is specified for LF1. If you use LF2 to add a record to PF1, you cannot add the record if it causes a duplicate key value in LF1.

The following shows the DDS for a logical file that requires unique key values:

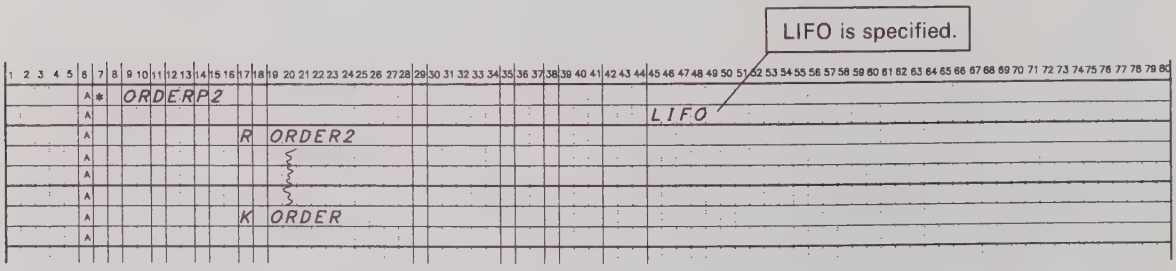
```
A*  ORDER TRANSACTION LOGICAL FILE (ORDFILL)
A                                  UNIQUE
A         R ORDHDR                 PFILE(ORDHDRP)
A         K ORDER
A
A         R ORDDTL                 PFILE(ORDDTLP)
A         K ORDER
A         K LINE
```

RSLH210-0

In this example, the contents of the key fields (the *Order* field for the ORDHDR record format, and the *Order* and *Line* fields for the ORDDTL record format) must be unique whether the record is added through: the ORDHDRP file , the ORDDTLP file, or the logical file defined here. With the *Line* field specified as a second key field in the ORDDTL record format, the same value can exist in the *Order* key field in both physical files. Because the physical file ORDDTLP has two key fields and the physical file ORDHDRP has only one, there is not a conflict between the key values in the two files.

Arranging Duplicate Keys: If you do not specify the UNIQUE keyword in DDS, then you may specify how the system is to store records with duplicate key values, should they occur. You specify that records with duplicate key values are stored in the access path in one of the following ways:

- Last-in-first-out (LIFO). When the LIFO keyword is specified, records with duplicate key values are retrieved in last-in-first-out order by the physical sequence of the records. Below, is an example of DDS using the LIFO keyword.



RSLH212-1

- First-in-first-out (FIFO). If the FIFO keyword is specified, records with duplicate key values are retrieved in first-in-first-out order by the physical sequence of the records.
- No specific order for duplicate key fields (the default). When neither the FIFO or LIFO keywords are specified, there is no guaranteed order for retrieving records with duplicate keys. No specific order for duplicate key fields allows more access path sharing, which can improve performance. For more information about access path sharing, see "Using Existing Access Paths" on page 6-13.

When a simple- or multiple-format logical file is based on more than one physical file member, records with duplicate key values are read in the order in which the files and members are specified on the DTAMBRS parameter on the Create Logical File (CRTLF) or Add Logical File Member (ADDLFM) command. For more information, see "Logical File Members" on page 6-19.

The LIFO or FIFO order of records with duplicate key values is not determined by the sequence of updates made to the contents of the key fields, but solely by the physical sequence of the records in the file member. Assume that a physical file has the FIFO keyword specified, (records with duplicate keys are in first-in-first-out order), and that the following shows the order in which records were added to the file:

Order Records Were Added to File	Key Value
1	A
2	B
3	C
4	C
5	D

The sequence of the access path is (FIFO, ascending key):

Record Number	Key Value
1	A
2	B
3	C
4	C
5	D

Records 3 and 4, which have duplicate key values, are in FIFO order. That is, because record 3 was added to the file before record 4, it is read before record 4. This would become apparent if the records were read in descending order. This could be done by creating a logical file based on this physical file, with the DESCEND keyword specified in the logical file.

The sequence of the access path is (FIFO, descending key):

Record Number	Key Value
5	D
3	C
4	C
2	B
1	A

If physical record 1 is changed such that the key value is C, the sequence of the access path for the physical file is (FIFO, ascending key):

Record Number	Key Value
2	B
1	C
3	C
4	C
5	D

Finally, changing to descending order, the new sequence of the access path for the logical file is (FIFO, descending key):

Record Number	Key Value
5	D
1	C
3	C
4	C
2	B

After the change, record 1 does not appear after record 4, even though the contents of the key field was updated after record 4 was added.

Using Existing Access Path Specifications

You can use the DDS keyword REFACCPTH to use another file's access path specifications. When the file is created, the system determines which access path to share. The file using the REFACCPTH keyword does not necessarily share the access path of the file specified in the REFACCPTH keyword. The REFACCPTH keyword is used to simply reduce the number of DDS statements that must be specified. That is, rather than code the key field specifications for the file, you can specify the REFACCPTH keyword. When the file is created, the system will copy the key field and select/omit specifications from the file specified on the REFACCPTH keyword.

Using Floating Point Fields in Access Paths

The collating sequence for records in a keyed data base file depends on the presence of the SIGNED, UNSIGNED, and ABSVAL DDS keywords. For floating-point fields, the sign is the leftmost bit, the exponent is next, and the significand is last. The collating sequence with UNSIGNED specified is:

- Positive real numbers—positive infinity
- Negative real numbers—negative infinity

A floating-point key field with the SIGNED keyword specified, or defaulted to, on the DDS has an algebraic numeric sequence. The collating sequence is negative infinity—real numbers—positive infinity.

A floating-point key field with the ABSVAL keyword specified on the DDS has an absolute value numeric sequence. The collating sequence is real numbers—infinities.

The following floating-point collating sequences are observed:

- Zero (positive or negative) collates in the same manner as any other positive/negative real number.
- Negative zero collates before positive zero for SIGNED sequences.
- Negative and positive zero collate the same for ABSVAL sequences.

You cannot use not-a-number (*NAN) values in key fields. If you attempt this, and a *NAN value is detected in a key field during file creation, the file will not be created.

Creating Data Base Files

The system supports several methods for creating a data base file:

- OS/400 IDDU
- Structured Query Language/400
- OS/400 control language (CL)

You can create a data base file using IDDU. If you are using IDDU to describe your data base files, you might also consider using it to create your files. For more information about IDDU, see the *IDDU User's Guide*.

You can create a data base file using the SQL/400 statements. SQL/400 is the IBM Systems Application Architecture relational data base language, and can be used on the AS/400 system to interactively describe and create data base files. For more information about SQL/400, see the *SQL/400 Programmer's Guide*.

You can also create a data base file by using CL. The CL data base file create commands are: Create Physical File (CRTPF), Create Logical File (CRTLF), and Create Source Physical File (CRTSRCPF).

Because there are additional system functions available with CL, this guide will focus on creating files using CL.

Specifying Data Base File and Member Attributes

When you create a data base file, data base attributes are stored with the file and members. Some of these file and member attributes are discussed in this section. For a complete discussion about how to specify these attributes and parameters, see the CRTPF, CRTLF, CRTSRCPF, ADDPFM, ADDLFM, CHGPF, CHGLF, CHGPFM, CHGSRCPF, and CHGLFM commands in the *CL Reference*.

File and Member Name

FILE and MBR Parameters. When you create a file, you identify the name of the file on the create command (on the FILE parameter). You also identify the library in which the file will reside. When you create a physical or logical file, the system normally creates a member by the same name as the file. You can, however, specify your own member name on the MBR parameter on the create commands. You can also tell the system not to create any members when the file is created by specifying MBR(*NONE) on the create commands.

Note: When you create a source physical file, the system does *not* automatically create a member.

You can control the order in which physical file members are read by using a logical file. You can use the DTAMBRS parameter of the Create Logical File (CRTLF) command to specify the order the physical file members are to be read. For more information about using logical files in this way, see "Logical File Members" on page 6-19.

Source File and Member

SRCFILE and SRCMBR Parameters. If the file being created is using DDS source statements to describe the file, the SRCFILE and SRCMBR parameters specify which source file and member, respectively, contains the DDS. If you do not specify a name, the default source file name is QDDSSRC, and the default member name is the same name you specified on the FILE parameter.

Type of Data Base File

FILETYPE Parameter. When you create a file, you can specify whether the file is a data file or a source file. The Create Physical File (CRTPF) and Create Logical File (CRTLF) commands default to a data file type (*DATA). The Create Source Physical File (CRTSRCPF) command defaults to a source file type (*SRC).

Number of Members Allowed in the File

MAXMBRS Parameter. When you create a file, you can also specify the maximum number of members the file can hold. The MAXMBRS parameter is used to control this value. The default maximum number of members for physical and logical files is one, and the default for source physical files is *NOMAX.

Where the Data Is to Be Stored

UNIT Parameter. You need not specify the location of data in auxiliary storage. The system automatically finds a place for the file on auxiliary storage. However, if you wish to specify which disk you want the file located on, specify the UNIT parameter. The UNIT parameter specifies the location of the data records in physical files and the access path for both physical files and logical files.

If there is not enough space on the unit, or if the unit specified is not valid for your system, the data and/or access paths are placed on different units. An informational message indicating that the file was not placed on the requested unit is sent at the time the file members are added. (A message is *not* sent when the file member is extended.)

In general, you should not specify the UNIT parameter. The suggested approach is to allow the system to place the file on the disk unit of its choosing. This is normally a better approach for performance, and also relieves you of the task of managing auxiliary storage.

Frequency of Writing Data to Auxiliary Storage

FRCRATIO Parameter. Normally, the system determines when to write changed data from main storage to auxiliary storage. If you want to control when data base changes are written to auxiliary storage, you can do so by using the force write ratio (FRCRATIO) parameter on either the create, change, or override data base file commands.

If you do not specify that records are to be forced to auxiliary storage, the system determines when the records are written. Closing the file (except for a shared close) and the force-end-of-data operation automatically forces remaining updates,

deletions, and additions to auxiliary storage. If you are journaling the file, the FRCRATIO parameter should normally be *NONE.

Using the FRCRATIO parameter has performance and recovery considerations for your system. To understand these considerations, see Chapter 16 in this guide, and the *Backup and Recovery Guide*.

Frequency of Writing the Access Path to Auxiliary Storage

FRCACCPTH Parameter. The force access path (FRCACCPTH) parameter controls when an access path is written to auxiliary storage.

Specifying FRCACCPTH(*YES) forces the access path to auxiliary storage whenever the access path is changed. This can reduce the chance that the access path will need to be rebuilt should the system fail because the system has a copy of the access path in auxiliary storage.

Specifying FRCACCPTH(*YES) can degrade performance when changes occur to the access path. An alternative to forcing the access path is journaling the access path. For more information about forcing access paths and journaling access paths, see Chapter 16 in this guide, and the *Backup and Recovery Guide*.

Checking for Changes to the Record Format Description

LVLCHK Parameter. The system checks, when the file is opened, if the definition of the data base file you are using was changed to an extent that your program may not be able to process the file. The system normally notifies your program of this condition. For example, assume you compiled your program two months ago and, at that time, the file the program was using was defined as having three fields in each record. Last week another programmer decided to add a new field to the record format, so that now each record would have four fields. The system will notify your program, when it tries to open the file, that a significant change occurred to the definition of the file since the last time the program was compiled. This notification is known as a record format level check. You can specify if you want level checking when you create a file or by using one of the change data base file commands (the default is to do level checking). You can override the system and ignore the level check by using the Override Data Base File (OVRDBF) command.

Keeping the Access Path Current

MAINT Parameter. While a file is open, the system maintains the access paths as changes are made to the data in the file. However, because more than one access path can exist for the same data, changing data in one file might cause changes to be made in access paths for other files that are not currently open (in use). There are three ways of maintaining access paths of files that are not currently open:

- Immediate
- Rebuild
- Delayed

Immediate maintenance of an access path means that the access path is maintained as changes are made to its associated data, regardless if the file is open.

Rebuild maintenance of an access path means that the access path is only maintained while the file is open, not when the file is closed; the access path is rebuilt when the file is opened the next time. When a file with rebuild maintenance is closed, the system stops maintaining the access path. When the file is opened again,

the access path is totally rebuilt. If one or more programs has opened a specific file member with rebuild maintenance specified, the system maintains the access path for that member until the last user closes the file member.

Delayed maintenance of an access path means that any maintenance for the access path is done after the file member is opened the next time and while it remains open. However, the access path is not rebuilt as it is with rebuild maintenance. Updates to the access path are collected from the time the member is closed until it is opened again. When it is opened, only the collected changes are merged into the access path.

Delayed maintenance can be faster than rebuilding the entire access path (as you would if you specify rebuild maintenance). However, if the collected changes to the access path exceed approximately 25% of the number of entries in the access path, the changes are no longer accumulated and the system rebuilds the access path at the next open (because the system can usually rebuild the entire access path faster than merging many collected changes into it). You should use delayed maintenance for files that have relatively few changes to the access path while the file members are closed. Delayed maintenance reduces system overhead by reducing the number of access paths that are maintained immediately. It also does not require the time needed to totally rebuild the access path when the member is opened.

The type of access path maintenance to specify depends on the number of records and the frequency of additions, deletions, and updates to a file while the file is closed. If you do not specify the type of maintenance for a file, the default is immediate maintenance.

Immediate maintenance decreases the open time, but increases the amount of time required to update records that affect changes to the access path while it is closed. You may want to specify immediate maintenance for access paths that are used frequently, or when you cannot wait for an access path to be rebuilt when the file is opened. You may want to specify delayed maintenance for access paths that are not used frequently, if there are infrequent changes to the record keys that make up the access path.

In general, for files used interactively, immediate maintenance results in good response time. For files used in batch jobs, either immediate, delayed, or rebuild maintenance is adequate, depending on the size of the members and the frequency of changes.

The following is a comparison of immediate, rebuild, and delayed maintenance as they affect opening and processing files:

Immediate	Rebuild	Delayed
Fast open because the access path is current.	Slow open because access path must be rebuilt.	Moderately fast open because the access path does not have to be rebuilt, but it must still be changed. Slow open if extensive changes are needed.
Slower update/output operations when many access paths with immediate maintenance are built over changing data (the system must maintain the access paths).	Faster update/output operations when many access paths with rebuild maintenance are built over changing data and are not open (the system does not have to maintain the access paths).	Moderately fast update/output operations when many access paths with delayed maintenance are built over changing data and are not open, (the system records the changes, but the access path itself is not maintained).

Notes:

1. Delayed or rebuild maintenance cannot be specified for a file that has unique keys.
2. Rebuild maintenance cannot be specified for a file if its access path is being journaled.

Recovering an Access Path If the System Fails

RECOVER Parameter. If the system fails, the access paths that had changes that were not forced to auxiliary storage when the system failed or that were not journaled, cannot be used until they are rebuilt. Using the RECOVER parameter on the Create Physical File (CRTPF), the Create Logical File (CRTLF), and the Create Source Physical File (CRTSRCPF) commands, you can specify when that access path is to be rebuilt. Access paths are rebuilt either during the initial program load (IPL), after the IPL, or when a file is opened.

The following shows the relationship among duplicate key options, maintenance options, and recovery options:

Duplicate Key Options	Maintenance Options	Recovery Options
Unique	Immediate	Rebuild during the IPL (*IPL)
		Rebuild after the IPL (*AFTIPL, default)
		Do not rebuild at IPL, wait for first open (*NO).
Not unique	Immediate or delayed	Rebuild during the IPL (*IPL)
		Rebuild after the IPL (*AFTIPL)
		Do not rebuild at IPL, wait for first open (*NO, default)
Not unique	Rebuild	Do not rebuild at IPL, wait for first open (*NO, default)

If access paths must be recovered, a list of those files whose access paths need to be recovered is shown on the Override Access Path Recovery display at the next IPL. At that time, you can override the recovery option that was originally specified for the file by taking the appropriate option on the display. For more information about this display, see the *Backup and Recovery Guide*.

Sharing a File in the Same Job

SHARE Parameter. By default, the data base system lets a file be accessed and changed at the same time by many users. The SHARE parameter allows even closer sharing by sharing opened files in the same job. For example, sharing a file in the same job allows programs in the same job to share a file's status, record position, and buffer. Sharing files in the same job can improve performance by reducing the amount of storage the job needs and by reducing the time it takes to open and close the file. For more information about sharing files in the same job, see "Sharing Data Base Files in the Same Job" on page 8-7.

Specifying Wait Time for a Locked File or Record

WAITFILE and WAITRCD Parameter. When you create a file, you can specify how long a program should wait for either the file or a record in the file if another job has the file or record locked. If the wait time ends before the file or record is released, a message is sent to the program indicating that the job was not able to use the file or read the record. For more information about record and file locks and wait times, see "Record Locks" on page 8-5 and "File Locks" on page 8-6.

Specifying Public Authority

AUT Parameter. When you create a file, you can specify public authority. Public authority is the authority a user has to a file (or other object on the system) if that user does not have specific authority for the file or does not belong to a group with specific authority for the file. For more information about public authority, see "Public Authority" on page 7-3.

Specifying The System on Which the File Is Created

SYSTEM Parameter. You can specify if the file is to be created on the local system or a remote system that supports DDM. For more information about DDM, see the *DDM User's Guide*.

Specifying File and Member Text

TEXT Parameter. You can specify a text description for each file and member you create. The text data is useful in documenting information about your file and members.

Chapter 5. Setting Up Physical Files

This chapter discusses some of the unique considerations for describing, then creating, a physical file.

For information about describing a physical file record format, see "Example of Describing a Physical File Using DDS" on page 4-7.

For information about describing a physical file access path, refer to "Describing the Access Path for the File" on page 4-16.

Creating a Physical File

To create a physical file, take the following steps:

1. If you are using DDS, enter DDS for the physical file into a source file. This can be done using the AS/400 Application Development Tools source entry utility (SEU). See "Working with Source Files" on page 17-3, for more information about how source statements are entered in source files.
2. Create the physical file. You can use the Create Physical File (CRTPF) command, or the Create Source Physical File (CRTSRCPF) command.

The following command creates a one member file using DDS and places it in a library called DSTPRODLB.

```
CRTPF FILE(DSTPRODLB/ORDHDRP)
      TEXT('Order header physical file')
```

As shown, this command uses defaults. For the SRCFILE and SRCMBR parameters, the system uses DDS in the source file called QDDSSRC and the member names ORDHDRP (the same as the file name). The file ORDHDRP with one member of the same name is placed in the library DSTPRODLB.

Specifying Physical File and Member Attributes

Some of the attributes you can specify for physical files on the Create Physical File (CRTPF), Create Source Physical File (CRTSRCPF), Change Physical File (CHGPF), Change Source Physical File (CHGSRCPF), Add Physical File Member (ADDPFM), and Change Physical File Member (CHGPFM) commands include (command names are given in parentheses):

Expiration Date

EXPDATE Parameter. This parameter specifies an expiration date for each member in the file (ADDPFM, CHGPFM, CRTPF, CHGPF, CRTSRCPF, and CHGSRCPF commands). If the expiration date is past, the system operator is notified when the file is opened. The system operator can then override the expiration date and continue, or stop the job. Each member can have a different expiration date, which is specified when the member is added to the file. (The expiration date check can be overridden; see "Checking for the File's Expiration Date" on page 8-5.)

Size of the Physical File Member

SIZE Parameter. This parameter specifies the maximum number of records that can be placed in each member (CRTPF, CHGPF, CRTSRCPF, AND CHGSRCPF commands). The following formula can be used to determine the maximum:

R + (I * N)

where:

R Is the starting record count
I Is the number of records (increment) to add each time
N Is the number of times to add the increment

The defaults for the SIZE parameter are:

R 10,000
I 1,000
N 3 (CRTPF command)
499 (CRTSRCPF command)

For example, assume that R is a file created for 5000 records plus 3 increments of 1000 records each. The system can add 1000 to the initial record count of 5000 three times to make the total maximum 8000. When the total maximum is reached, the system operator either stops the job or tells the system to add another increment of records and continue. When increments are added, a message is sent to the system history log.

Instead of taking the default size or specifying a size, you can specify *NOMAX. For information about the maximum number of records allowed in a file, see Appendix A.

Storage Allocation

ALLOCATE Parameter. This parameter controls the storage allocated for members when they are added to the file (CRTPF, CHGPF, CRTSRCPF, and CHGSRCPF commands). The storage allocated would be large enough to contain the initial record count for a member. If you do not allocate storage when the members are added, the system will automatically extend the storage allocation as needed. You can use the ALLOCATE parameter only if you specified a maximum size on the SIZE parameter. If SIZE(*NOMAX) is specified, then ALLOCATE(*YES) cannot be specified.

Method of Allocating Storage

CONTIG Parameter. This parameter controls the method of allocating physical storage for a member (CRTPF and CRTSRCPF commands). If you allocate storage, you can request that the storage for the starting record count for a member be contiguous. That is, all the records in a member are to physically reside together. If there is not enough contiguous storage, contiguous storage allocation is not used and an informational message is sent to the job that requests the allocation, at the time the member is added.

Note: When a physical file is first created, the system always tries to allocate its initial storage contiguously. The only difference between using CONTIG(*NO) and CONTIG(*YES) is that with CONTIG(*YES) the system sends a message to the job log if it is unable to allocate contiguous storage when the file is created. No message is sent when a file is extended regardless of what you specified on the CONTIG parameter.

Record Length

RCDLEN Parameter. This parameter specifies the length of records in the file (CRTPF and CRTSRCPF commands). If the file is described to the record level only, then you specify the RCDLEN parameter when the file is created. This parameter cannot be specified if the file is described using DDS, IDDU, or SQL/400 (the system automatically determines the length of records in the file from the field level descriptions).

Deleted Records

DLTPCT Parameter. This parameter specifies the percentage of deleted records a file can contain before you want the system to send a message to the system history log (CRTPF, CHGPF, CRTSRCPF, and CHGSRCPF commands). When a file is closed, the system checks the member to determine the percentage of deleted records. If the percentage exceeds that value specified in the DLTPCT parameter, a message is sent to the history log. (For information about processing the history log, see the chapter on message handling in the *CL Programmer's Guide*.) One reason you might want to know when a file reaches a certain percentage of deleted records is to reclaim the space used by the deleted records. After you receive the message about deleted records, you could run the Reorganize Physical File Member (RGZPFM) command to reclaim the space. (For more information about RGZPFM, see "Reorganizing Data in Physical File Members" on page 13-3.) You can also specify to bypass the deleted records check by using the *NONE value for the DLTPCT parameter. *NONE is the default for the DLTPCT parameter.

Physical File Capabilities

ALWUPD and ALWDLT Parameters. File capabilities are used to control which input/output operations are allowed for a data base file independent of data base authority. For more information about data base file capabilities and authority, see Chapter 7.

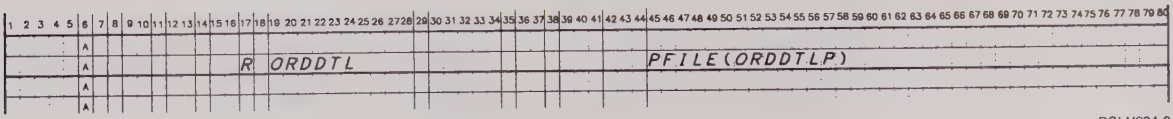
Chapter 6. Setting Up Logical Files

This chapter discusses some of the unique considerations for describing, then creating, a logical file. Many of the rules for setting up logical files apply to all categories of logical files. In this guide, rules that apply only to one category of logical file identify which category they refer to. Rules that apply to all categories of logical files do not identify specifically which category of logical file they apply to.

Describing Logical File Record Formats

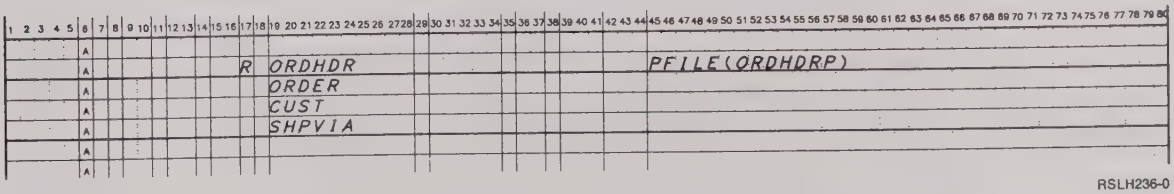
For every logical file record format described with DDS, you must specify a record format name and either the PFILE keyword (for simple and multiple format logical files), or the JFILE keyword (for join logical files). The file names specified on the PFILE or JFILE keyword are the physical files that the logical file is based on. A simple or multiple-format logical file record format can be specified with DDS in any one of the following ways:

1. In the simple logical file record format, specify only the record format name and the PFILE keyword. The record format for the only (or first) physical file specified on the PFILE keyword is the record format for the logical file. The record format name specified in the logical file must be the same as the record format name in the only (or first) physical file.



RSLH234-0

2. In this example, you describe your own record format by listing the field names you want to include. You can specify the field names in a different order, you can rename fields using the RENAME keyword, you can combine fields using the CONCAT keyword, and you can use specific positions of a field using the SST keyword. You can also override attributes of the fields by specifying the attributes again in the logical file.



RSLH236-0

3. In this example, the parameter value specified on the FORMAT keyword is the name of a data base file from which the only (or first) record format is to be copied for the logical file record format being described. The physical file name can be qualified by a library name. If a library name is not specified, the library list is used to find the physical file. The physical file must exist when the file you are describing is created. In addition, the record format name you specify in the logical file must be the same as the record format name in the file you specify on the FORMAT keyword.

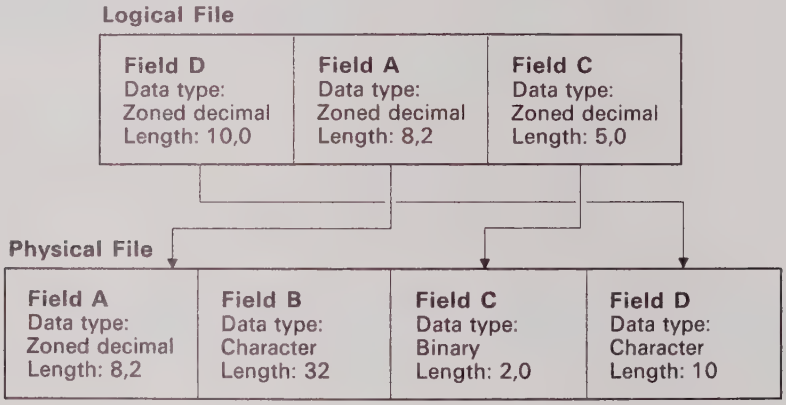
```
     A
     A                R CUSRCD                    PFILE(CUSMSTP)
     A                                            FORMAT(CUSMSTL)
     A
     A
```

RSLH235-0

In the following example, a program needs:

- The fields placed in a different order.
- A subset of the fields from the physical file.
- The data types changed for some fields.
- The field lengths changed for some fields.

You can use a logical file to make these changes.



RSLH205-1

For the logical file, the DDS would be:

```
     A                R LOGREC                    PFILE(PF1)
     A                  D          10S 0
     A                  A
     A                  C           5S 0
     A
     A
```

RSLH206-0

For the physical file, the DDS would be:

```
     A                R PHYREC
     A                  A          8S 2
     A                  B         32
     A                  C          2B 0
     A                  D         10
     A
     A
```

RSLH207-0

When a record is read from the logical file, the fields from the physical file are changed to match the logical file description. If the program updates or adds a record, the fields are changed back. (For an add or update operation using a logical

file, the program must supply data that conforms with the format used by the logical file.)

The following chart shows what types of data mapping are valid between physical and logical files.

Physical File Data Type	Logical File Data Type				
	Character	**Zoned**	**Packed**	**Binary**	**Floating Point**
Character	Valid	See Note 1	Not valid	Not valid	Not valid
Zoned	See Note 1	Valid	Valid	See Note 2	Valid
Packed	Not valid	Valid	Valid	See Note 2	Valid
Binary	Not valid	See Note 2	See Note 2	See Note 3	See Note 2
Floating Point	Not valid	Valid	Valid	See Note 2	Valid

Notes:

1. Valid only if the number of characters equals the number of digits.
2. Valid only if the binary field has zero decimal positions.
3. Valid only if both binary fields have the same number of decimal positions.

Note: For information about mapping DBCS fields, see Appendix B.

Describing Field Usage for Logical Files

You can specify that fields in data base files are to be input-only, both (input/output), or neither fields. Do this by specifying one of the following in position 38:

Entry	Meaning
Blank	For simple or multiple format logical files, defaults to B (both) For join logical files, defaults to I (input only)
B	Both input and output allowed; not valid for join logical files
I	Input only (read only)
N	Neither input nor output; valid only for join logical files

Note: Entries in position 38 cannot be referred to by the REF or REFFLD keywords.

Both

A both field can be used for both input and output operations. Your program can read data from the field and write data to the field. Both fields are not valid for join logical files, because join logical files are read-only files.

Input Only

An input only field can be used for read operations only. Your program can read data from the field, but cannot update the field in the file. Typical cases of input-only fields are key fields (to reduce maintenance of access paths by preventing changes to key field values), sensitive fields that a user can see but not update (for example, salary), and fields for which the translate table (TRNTBL) keyword is specified (for an example, see the TRNTBL keyword in the *DDS Reference*).

If your program updates a record in which you have specified input-only fields, the input-only fields are not changed in the file. If your program adds a record that has input-only fields, the input-only fields take default values (see the DFT keyword in the *DDS Reference*).

Neither

A neither field is used neither for input or output. It is valid only for join logical files. A neither field can be used as a join field in a join logical file, but your program cannot read or update a neither field.

Use neither fields when the attributes of join fields in the physical files do not match. In this case, one or both join fields must be defined again. However, you may not want to include these redefined fields in the record format (you may not want the application program to see the redefined fields.) Therefore, redefined join fields can be coded N so that they do not appear in the record format.

A field with N in position 38 does not appear in the buffer used by your program. However, the field description is displayed with the Display File Field Description (DSPFFD) command.

Neither fields cannot be used as select/omit or key fields.

For an example of a neither field, see "Describing Fields That Never Appear in the Record Format (Example 5)" on page 6-36.

Deriving New Fields from Existing Fields

Fields in a logical file can be derived from fields in the physical file the logical file is based on or from fields in the same logical file. For example, you can concatenate two or more fields from a physical file to make them appear as one field in the logical file. Likewise, you can divide one field in the physical file to make it appear as multiple fields in the logical file.

Concatenated Fields

Using the CONCAT keyword, you can combine two or more fields from a physical file record format to make one field in a logical file record format. For example, a physical file record format contains the fields *Month*, *Day*, and *Year*. For a logical file, you concatenate these fields into one field, *Date*.

The field length for the resulting concatenated field is the sum of the lengths of the included fields (unless the fields in the physical file are binary or packed decimal, in which case they are changed to zoned decimal). The field length of the resulting field is automatically calculated by the system. A concatenated field can have:

- Column headings
- Validity checking
- Text description
- Edit code or edit word (numeric concatenated fields only)

Note: This editing and validity checking information is not used by the data base management system but is retrieved when field descriptions from the data base file are referred to in a display or printer file.

When fields are concatenated, the data types can change (the resulting data type is automatically determined by the system). The following table shows what data types result:

	Any Numeric	Character
Any Numeric	Zoned decimal	Character
Character	Character	Character

Note: For information about concatenating DBCS fields, see Appendix B.

When only numeric fields are concatenated, the sign of the last field in the group is used as the sign of the concatenated field.

Notes:

1. Numeric fields with decimal precision other than zero cannot be included in a concatenated field.
2. Floating-point fields cannot be included in a concatenated field.
3. For join logical files, all fields specified as parameter values on the CONCAT keyword must be from the same physical file. (The first field must identify the physical file it comes from by its uniqueness or from the JREF keyword.)

The following shows the field description in DDS for concatenation. (The CONCAT keyword is used to specify the fields to concatenate.)

```
00101 A          MONTH
00102 A          DAY
00103 A          YEAR
00104 A          DATE                 CONCAT(MONTH DAY YEAR)
```

RSLH247-0

In this example, the logical file record format includes the separate fields of *Month*, *Day*, and *Year*, as well as the concatenated *Date* field. Any of the following can be used:

- A format with the separate fields of *Month*, *Day*, and *Year*
- A format with only the concatenated *Date* field
- A format with the separate fields *Month*, *Day*, *Year* and the concatenated *Date* field

When both separate and concatenated fields exist in the format, any updates to the fields are processed in the sequence in which the DDS is specified. In the previous example, if the *Date* field contained 103188 and the *Month* field is changed to 12, when the record is updated, the month in the *Date* field would be used. The updated record would contain 103188. If the *Date* field were specified first, the updated record would contain 123188.

Concatenated fields can also be used as key fields and select/omit fields.

Substring Fields

You can use the SST keyword to substring a character or zoned decimal field. For example, assume you had defined the *Date* field in your physical file as 6 characters in length. You can describe the logical file with three fields, each 2 characters in length. You could use the SST keyword to define MM as 2 characters starting in position 1 of the *Date* field, DD as 2 characters starting in position 3 of the *Date* field, and YY as 2 characters starting in position 5 of the *Date* field.

Substring fields can also be used as key fields and select/omit fields.

Renamed Fields

You can name a field in a logical file differently than its name in the physical file. You might want to rename a field in a logical file because the program was written using a different field name or because the original field name does not conform to the naming restrictions of the high-level language you are using.

Translated Fields

You can specify a translate table for a field using the DDS keyword, TRNTBL. When you read a logical file record and a translate table was specified for one or more fields in the logical file, the system translates the data from the field value in the physical file to the value determined by the translate table.

Describing Floating-Point Fields in Logical Files

You can use floating-point fields as mapped fields in logical files. A single- or double-precision floating-point field can be mapped to or from a zoned, packed, zero-precision binary, or another floating-point field. You cannot map between a floating-point field and a nonzero-precision binary field, a character field, or a DBCS field when you are creating a file.

Mapping between floating-point fields of different precision, single or double, or between floating-point fields and other numeric fields, may result in rounding or a loss of precision. Mapping a double-precision floating-point number to a single-precision floating-point number may result in rounding, depending on the particular number involved and its internal representation. Rounding is to the nearest (even) bit. The result always contains as much precision as possible. A loss of precision can also occur between two decimal numbers if the number of digits of precision is decreased.

You can inadvertently change the value of a field which your program did not explicitly change. For floating-point fields, this can occur if a physical file has a double-precision field that is mapped to a single-precision field in a logical file, and you issue an update for the record through the logical file. If the internal representation of the floating-point number causes it to be rounded when it is mapped to the logical file, then the update of the logical record causes a permanent loss of precision in the physical file. If the rounded number is the key of the physical record, then the sequence of records in the physical file may also change.

A fixed-point numeric field can also be updated inadvertently if the precision is decreased in the logical file.

Describing Access Paths for Logical Files

There are three ways of specifying the access path for a logical file record format.

1. Keyed sequence access path specification. Specify key fields after the last record or field level specification. The key field names must be in the record format. For join logical files, the key fields must come from the first, or primary, physical file.

```
A                R CUSRCD                    PFILE(CUSMSTP)
A                K ARBAL
A                K CRDLMT
A
```

RSLH237-1

2. Arrival sequence access path specification. Specify no key fields. You can specify only one physical file on the PFILE keyword (and only one of the physical file's members when you add the logical file member).

```
A                R CUSRCD                    PFILE(CUSMSTP)
A
A
```

RSLH239-1

3. Previously defined keyed sequence access path specification (for simple and multiple format logical files only). Specify the REFACCPTH keyword at the file level to identify a previously created data base file whose access path and select/omit specifications this logical file is to refer to. You cannot specify individual key or select/omit fields with the REFACCPTH keyword.

   Note: Even though the specified file's access path specifications are used, the system determines which file's access path, if any, will actually be shared. The system always tries to share access paths, regardless of whether the REFACCPTH keyword is used.

```
A                                            REFACCPTH(DSTPRODLIB/ORDHDRL)
A                R CUSRCD                    PFILE(CUSMSTP)
A
```

RSLH238-2

Selecting and Omitting Records Using Logical Files

The system can select and omit records when using a logical file. This can help you to exclude records in a file for processing convenience or for security.

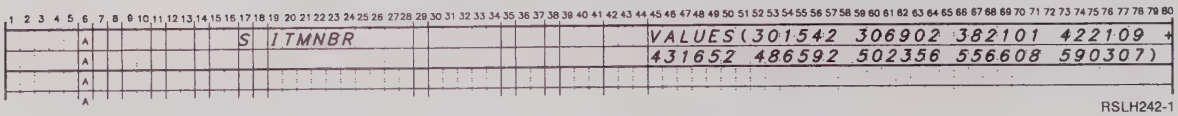
The process of selecting and omitting records is based on comparisons identified in position 17 of the *DDS Coding Form* for the logical file, and is similar to a series of comparisons coded in a high-level language program. For example, in a logical file that contains order detail records, you can specify that the only records you want to use are those in which the quantity ordered is greater than the quantity shipped. All other records are omitted from the access path. The omitted records remain in the physical file but are not retrieved for the logical file. If you are adding records to the physical file, all records are added but only the selected records are added to the access path.

In DDS, to specify select or omit, you specify an S (select) or O (omit) in position 17 of the *DDS Coding Form*. You then name the field (in positions 19 through 28) that will be used in the selection or omission process. In positions 45 through 80 you specify the comparison.

Note: Select/omit specifications appear after key specifications (if keys are specified).

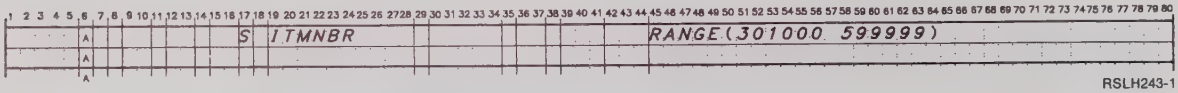
Records can be selected and omitted by several types of comparisons:

- VALUES. The contents of the field are compared to a list of not more than 100 values. If a match is found, the record is selected or omitted. In the following example, a record is selected if one of the values specified in the VALUES keyword is found in the *Itmnbr* field.



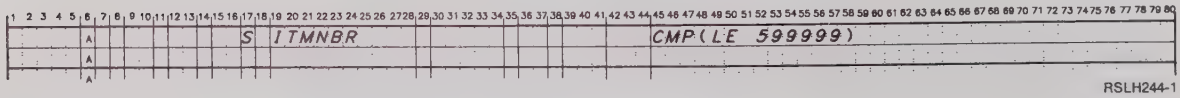
RSLH242-1

- RANGE. The contents of the field are compared to lower and upper limits. If the contents are greater than or equal to the lower limit and less than or equal to the upper limit, the record is selected or omitted. In the following example, all records with a range 301000 through 599999 in the *Itmnbr* field are selected.



RSLH243-1

- CMP. The contents of a field are compared to a value or the contents of another field. Valid comparison codes are EQ, NE, LT, NL, GT, NG, LE, and GE. If the comparison is met, the record is selected or omitted. In the following example, a record is selected if its *Itmnbr* field is less than or equal to 599999:



RSLH244-1

The value for a numeric field for which the CMP, VALUES, or RANGE keyword is specified is aligned based on the decimal positions specified for the field and filled with zeros where necessary. If decimal positions were not specified for the field, the decimal point is placed to the right of the rightmost digit in the value. For example, for a numeric field with length 5 and decimal positions 2, the value 1.2 is interpreted as 001.20 and the value 100 is interpreted as 100.00.

The status of a record is determined by evaluating select/omit statements in the sequence you specify them. If a record qualifies for selection or omission, subsequent statements are ignored.

Normally the select and omit comparisons are treated as independent from one another; the comparisons are ORed together. That is, if the select or omit comparison is met, the record is either selected or omitted. If the condition is not met, the system proceeds to the next comparison. To connect comparisons together, you simply leave a space in position 17 of the *DDS Coding Form*. Then, all the compar-

isons that were connected in this fashion must be met before the record is selected or omitted. That is, the comparisons are ANDed together.

The fewer comparisons, the more efficient the task is. So, when you have several select/omit comparisons, try to specify the one that selects or omits the most records first.

In the following examples, few records exist for which the *Rep* field is JSMITH. The examples show how to use DDS to select all the records before 1988 for a sales representative named JSMITH in the state of New York. All give the same results with different efficiency (in this example, **3** is the most efficient).

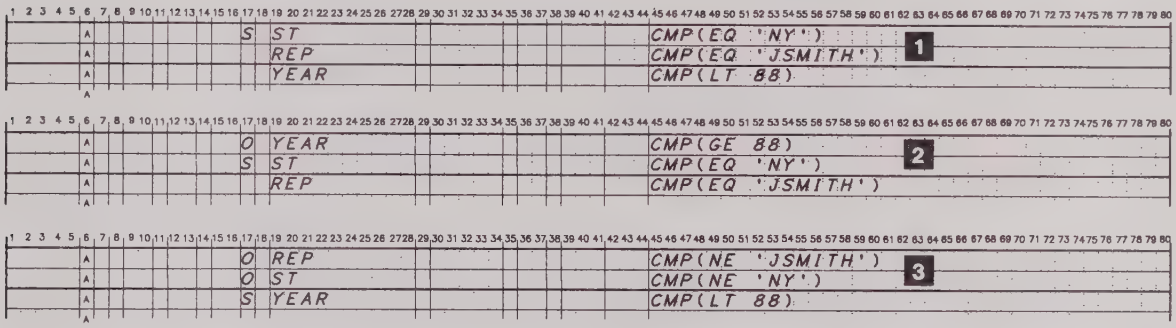


Figure 6-1. Three Ways to Code Select/Omit Function

1 All records must be compared with all of the select fields *St*, *Rep*, and *Year* before they can be selected or omitted.

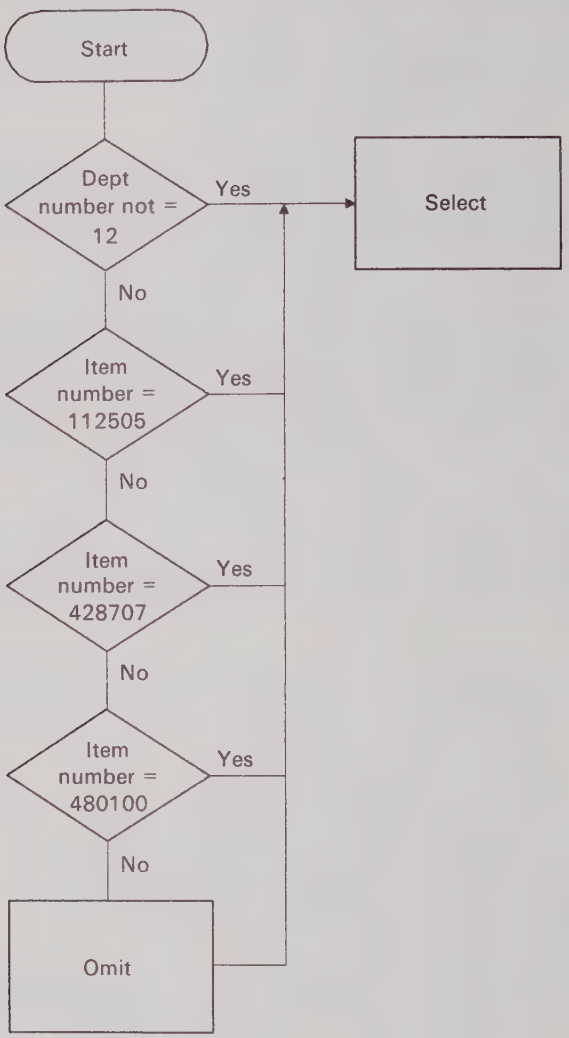
2 All records are compared with the *Year* field. Then, the records before 1988 have to be compared with the *St* and *Rep* fields.

3 All records are compared with the *Rep* field. Then, only the few for JSMITH are compared with the *St* field. Then, the few records that are left are compared to the *Year* field.

As another example, assume that you want to select the following:

- All records for departments other than Department 12.
- Only those records for Department 12 that contain an item number 112505, 428707, or 480100. No other records for Department 12 are to be selected.

The following diagram shows the logic included in this example:



RSLH245-0

The following shows how to code this example using the DDS select and omit functions:

```
A            S  DPTNBR        CMP(NE  12)
A            S  ITMNBR        VALUES(112505  428707  480100)
A
A
```

RSLH246-1

It is possible to have an access path with select/omit values and process the file in arrival sequence. For example, a high-level language program may specify that the keyed access path is to be ignored. In this case, every record is read from the file in arrival sequence, but only those records meeting the select/omit values specified in the file are returned to the high-level language program.

A logical file with key fields and select/omit values specified can be processed in arrival sequence or by using relative record numbers randomly. Records omitted by the select/omit values are not processed. That is, if an omitted record is requested by relative record number, the record is not returned to the high-level language program.

The system does not ensure that any additions or changes through a logical file will allow the record to be accessed again in the same logical file. For example, if the selection criteria of the logical file specifies only records with an A in the *Fld1* and the program updates the record with a B in *Fld1*, the program cannot retrieve the record again by using this logical file.

Note: You cannot select or omit based on the values of a floating-point field.

There are two kinds of select/omit operations: access path select/omit and dynamic select/omit. The default is access path select/omit. The select/omit specifications themselves are the same in each kind, but the system actually does the work of selecting and omitting records at different times.

Access Path Select/Omit

With access path select/omit, the access path only contains keys that meet the select/omit values specified for the logical file. When you specify key fields for a file, an access path is kept for the file and maintained by the system when you add or update records in the physical file(s) used by the logical file. The only index entries in the access path are those that meet the select/omit values.

Dynamic Select/Omit

With dynamic select/omit, when a program reads records from the file, the system only returns those records that meet the select/omit values. That is, the actual select/omit processing is done when records are read by a program, rather than when the records are added or changed. However, the keyed access path, contains all the keys, not just keys from selected records. Access paths using dynamic select/omit allow more access path sharing, which can improve performance. For more information about access path sharing, see "Using Existing Access Paths" on page 6-13.

To specify dynamic select/omit, use the dynamic selection (DYNSLT) keyword. With dynamic select/omit, key fields are not required.

If you have a file that is updated frequently and read infrequently, you may not need to update the access path for select/omit purposes until your program reads the file. In this case, dynamic select/omit might be the correct choice. The following example helps describe this.

You use a code field (A = active, I = inactive), which is changed infrequently, to select/omit records. Your program processes the active records and the majority (over 80%) of the records are active. It may be more efficient to use DYNSLT to dynamically select records at processing time rather than perform access path maintenance when the code field is changed.

Using the Open Query File Command to Select/Omit Records

Another method of selecting records is using the QRYSLT parameter on the Open Query File (OPNQRYF) command. The open data path created by the OPNQRYF command is like a temporary logical file; that is, it is automatically deleted when it is closed. A logical file, on the other hand, remains in existence until you specifically delete it. For more details about the OPNQRYF command, see "Using the Open Query File (OPNQRYF) Command" on page 9-2.

Using Existing Access Paths

When two or more files are based on the same physical files and the same key fields in the same order, they will automatically share the same keyed sequence access path. When access paths are shared, the amount of system activity required to maintain access paths and the amount of auxiliary storage used by the files is reduced.

When a logical file with a keyed sequence access path is created, the system always tries to share an existing access path. For access path sharing to occur, an access path must exist on the system that satisfies the following conditions:

- The logical file member to be added must be based on the same physical file members that the existing access path is based on.
- The length, data type, and number of decimal positions specified for each key field, must be identical in both the new file and the existing file.
- If neither the FIFO nor LIFO keyword is specified, then the new file can have fewer keys fields than the existing access paths. That is, a new logical file can share an existing access path if the beginning part of the key is identical.
- The attributes of the access path (such as UNIQUE, LIFO, or FIFO) and the attributes of the key fields (such as DESCEND, ABSVAL, UNSIGNED, and SIGNED) must be identical.

  Exceptions:
  1. A FIFO access path can share an access path in which the UNIQUE keyword is specified if all the other requirements for access path sharing are met.
  2. A UNIQUE access path can share a FIFO access path that needs to be rebuilt (for example, has *REBLD maintenance specified), if all the other requirements for access path sharing are met.
- If the new logical file has select/omit specifications, they must be identical to the select/omit specifications of the existing access path. However, if the new logical file specifies DYNSLT, it can share an existing access path if the existing access path has either:
  - The dynamic select (DYNSLT) keyword specified
  - No select/omit keywords specified
- The alternative collating sequence (ALTSEQ keyword) and the translation table (TRNTBL keyword) of the new logical file member, if any, must be identical to the alternative collating sequence and translation table of the existing access path.

Note: Logical files that contain concatenated or substring fields cannot share access paths with physical files.

The owner of any access path is the logical file member that originally created the access path. For a shared access path, if the logical member owning the access path is deleted, the first member to share the access path becomes the new owner. The FRCACCPTH, MAINT, RECOVER, and UNIT parameters on the Create Logical File (CRTLF) command need not match the same parameters on an existing access path for that access path to be shared. When an access path is shared by several logical

file members, and the FRCACCPTH, MAINT, RECOVER, and UNIT parameters are not identical, the system maintains the access path by the most restrictive value for each of the parameters specified by the sharing members. The following illustrates how this occurs:

MBRA specifies the following:	MBRB specifies the following:	System does the following:
FRCACCPTH (*NO) MAINT (*IMMED) RECOVER (*AFTIPL)	FRCACCPTH (*YES) MAINT (*DLY) RECOVER (*NO)	FRCACCPTH (*YES) MAINT (*IMMED) RECOVER (*AFTIPL)

RSLH222-2

The UNIT parameter for the access path is always taken from the owning logical member.

Access path sharing creates no dependencies between sharing members; therefore, it does not restrict the order in which members can be deleted.

The Display File Description (DSPFD) and Display Data Base Relations (DSPDBR) commands show access path sharing relationships.

Creating a Logical File

Before creating a logical file, the physical file or files on which the logical file is based must already exist.

To create a logical file, take the following steps:

1. Type the DDS for the logical file into a source file. This can be done using SEU or another method. See "Working with Source Files" on page 17-3, for how source is placed in source files. The following shows the DDS for logical file ORDHDRL (an order header file):

```
A*  ORDER HEADER LOGICAL FILE (ORDHDRL)
A            R ORDHDR                   PFILE(ORDHDRP)
A            K ORDER
A
A
```

RSLH248-0

   This file uses the key field *Order* (order number) to define the access path. The record format is the same as the associated physical file ORDHDRP. The record format name for the logical file must be the same as the record format name in the physical file because no field descriptions are given.

2. Create the logical file. You can use the Create Logical File (CRTLF) command.

   The following shows how the CRTLF command could be typed:

```
CRTLF  FILE(DSTPRODLB/ORDHDRL)
       TEXT('Order header logical file')
```

As shown, this command uses some defaults. For example, because the SRCFILE and SRCMBR parameters are not specified, the system used DDS from the IBM-supplied source file QDDSSRC, and the source file member name is ORDHDRL (the same as the

file name specified on the CRTLF command). The file ORDHDRL with one member of the same name is placed in the library DSTPRODLB.

Creating a Logical File with More Than One Record Format

A multiple format logical file lets you use related records from two or more physical files by referring to only one logical file. Each record format is always associated with one or more physical files. You can use the same physical file in more than one record format.

The following example shows how to create a logical file ORDFILL with two record formats. One record format is defined for order header records from the physical file ORDHDRP; the other is defined for order detail records from the physical file ORDDTLP. (Figure 6-2 on page 6-15 shows the DDS for the physical file ORDDTLP, Figure 6-3 on page 6-16 shows the DDS for the physical file ORDHDRP, and Figure 6-4 on page 6-16 shows the DDS for the logical file ORDFILL.)

The logical file record format ORDHDR uses one key field, *Order*, for sequencing; the logical file record format ORDDTL uses two keys fields, *Order* and *Line*, for sequencing.

```
A*  ORDER DETAIL FILE (ORDDTLP) - PHYSICAL FILE RECORD DEFINITION
A                                 REF(DSTREF)
A          R ORDDTL               TEXT('Order detail record')
A            CUST          R
A            ORDER         R
A            LINE          R
A            ITEM          R
A            QTYORD        R
A            DESCRP        R
A            PRICE         R
A            EXTENS        R
A            WHSLOC        R
A            ORDATE        R
A            CUTYPE        R
A            STATE         R
A            ACTMTH        R
A            ACTYR         R
A
A
```

RSLH249-0

Figure 6-2. DDS for Physical File ORDDTLP

```
A*  ORDER HEADER FILE (ORDHDRP) - PHYSICAL FILE RECORD DEFINITION
A                                                 REF(DSTREFP)
A          R ORDHDR                               TEXT('Order header record')
A            CUST              R
A            ORDER             R
A            ORDATE            R
A            CUSORD            R
A            SHPVIA            R
A            ORDSTS            R
A            OPRNME            R
A            ORDAMT            R
A            CUTYPE            R
A            INVNBR            R
A            PRTDAT            R
A            SEQNBR            R
A            OPNSTS            R
A            LINES             R
A            ACTMTH            R
A            ACTYR             R
A            STATE             R
```

RSLH228-1

Figure 6-3. DDS for a Physical File (ORDHDRP) Built From a Field Reference File

```
A*  ORDER TRANSACTION LOGICAL FILE (ORDFILL)
A          R ORDHDR                PFILE(ORDHDRP)
A          K ORDER
A
A          R ORDDTL                PFILE(ORDDTLP)
A          K ORDER
A          K LINE
A
A
```

RSLH250-0

Figure 6-4. DDS for the Logical File ORDFILL

To create the logical file ORDFILL with two associated physical files, use a Create Logical File (CRTLF) command like the following:

```
CRTLF  FILE(DSTPRODLB/ORDFILL)
       TEXT('Order transaction logical file')
```

The DDS source is in the member ORDFILL in the file QDDSSRC. The file ORDFILL with a member of the same name is placed in the DSTPRODLB library. The access path for the logical file member ORDFILL arranges records from both the ORDHDRP and ORDDTLP files. Record formats for both physical files are keyed on *Order* as the common field, and because of the order in which they were specified in the logical file description, they are merged in *Order* sequence with duplicates between files retrieved first from the header file ORDHDRP and second from the detail file ORDDTLP. Because neither FIFO or LIFO is specified, the order of retrieval of duplicate keys in the same file is not guaranteed.

Controlling How Records Are Retrieved in a File with Multiple Formats

In a logical file with more than one record format, key field definitions are required. Each record format has its own key definition, and the record format key fields can be defined to merge the records of the different formats. Each record format does not have to contain every key field in the key. Consider the following records:

Header Record Format:

Record	Order	Cust	Ordate
1	41882	41394	050688
2	32133	28674	060288

Detail Record Format:

Record	Order	Line	Item	Qtyord	Extens
A	32133	01	46412	25	125000
B	32133	03	12481	4	001000
C	41882	02	46412	10	050000
D	32133	02	14201	110	454500
E	41882	01	08265	40	008000

In DDS, the header record format is defined before the detail record format. If the access path uses the *Order* field as the first key field for both record formats and the *Line* field as the second key field for only the second record format, both in ascending sequence, the order of the records in the access path is:

Record 2
Record A
Record D
Record B
Record 1
Record E
Record C

Note: Records with duplicate key values are arranged first in the sequence in which the physical files are specified. Then, if duplicates still exist within a record format, the duplicate records are arranged in the order specified by the FIFO or LIFO keyword. For example, if the logical file specified the DDS keyword FIFO, then duplicate records within the format would be presented in first-in-first-out sequence.

For logical files with more than one record format, you can use the *NONE DDS function for key fields to separate records of one record format from records of other record formats in the same access path. Generally, records from all record formats are merged based on key values. However, if *NONE is specified in DDS for a key field, only the records with key fields that appear in all record formats before the *NONE are merged.

The logical file in the following example contains three record formats, each associated with a different physical file:

Record Format	Physical File	Key Fields
EMPMSTR	EMPMSTR	Empnbr (employee number)
EMPHIST	EMPHIST	Empnbr, Empdat (employed date)
EMPEDUC	EMPEDUC	Empnbr, Clsnbr (class number)

Note: All record formats have one key field in common, the *Empnbr* field.

The DDS for this example is:

```
A                K EMPNBR        } For EMPMSTR
A
A                K EMPNBR        }
A                K EMPDAT        } For EMPHIST
A
A                K EMPNBR        }
A                K *NONE         } For EMPEDUC
A                K CLSNBR        }
A
A
```

RSLH223-0

*NONE is assumed for the second and third key fields for EMPMSTR and the third key field for EMPHIST because no key fields follow these key field positions.

The following shows the arrangement of the records:

Empnbr	Empdat	Clsnbr	Record Format Name
426			EMPMSTR
426	6/15/74		EMPHIST
426		412	EMPEDUC
426		520	EMPEDUC
427			EMPMSTR
427	9/30/75		EMPHIST
427		412	EMPEDUC

*NONE serves as a separator for the record formats EMPHIST and EMPEDUC. All the records for EMPHIST with the same *Empnbr* field are grouped together and sorted by the *Empdat* field. All the records for EMPEDUC with the same *Empnbr* field are grouped together and sorted by the *Clsnbr* field.

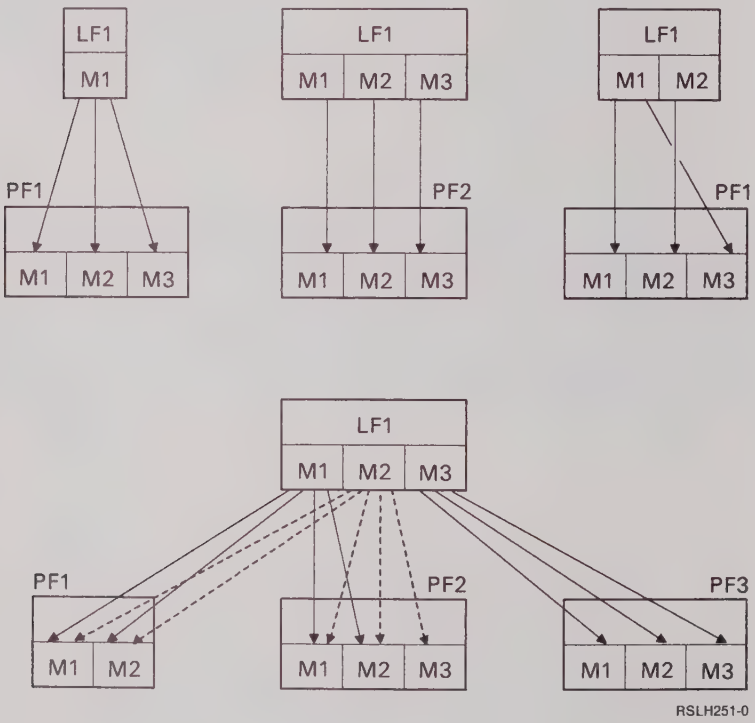
Controlling How Records Are Added to a File with Multiple Formats

When you add records to a multiple-format logical file and your application program uses a file name instead of a record format name, you need to write a format selector program. For more information about format selector programs, see "Identifying Which Record Format to Add in a File with Multiple Formats" on page 10-7.

Logical File Members

You can define members in logical files to separate the data into logical groups. The logical file member can be associated with one physical file member or with several physical file members.

The following illustrates this concept:



The record formats used with all logical members in a logical file must be defined in DDS when the file is created. If new record formats are needed, another logical file or record format must be created.

The attributes of an access path are determined by information specified in DDS and on commands when the logical file is created. The selection of data members is specified in the DTAMBRS parameter on the Create Logical File (CRTLF) and Add Logical File Member (ADDLFM) commands.

When a logical file is defined, the physical files used by the logical file are specified in DDS by the record level PFILE or JFILE keyword. If multiple record formats are defined in DDS, a PFILE keyword must be specified for each record format. You can specify one or more physical files for each PFILE keyword.

When a logical file is created or a member is added to the file, you can use the DTAMBRS parameter on the Create Logical File (CRTLF) or the Add Logical File Member (ADDLFM) command to specify which members of the physical files used by the logical file are to be used for data. Each physical file specified by a PFILE keyword in the DDS for the file should be specified on the DTAMBRS parameter. If no data members are to be selected from a physical file, that file should be specified on the DTAMBRS parameter with *NONE specified for the member.

In the following example, the logical file has two record formats defined:

```
0001.00 A                R LOGRCD2
                           PFILE(PF1 PF2)
0002.00 A                R LOGRCD3
                           PFILE(PF1 PF2 PF3)
```

RSLH252-0

The DTAMBRS parameter should specify the members to be selected from each of the physical files. The files should be specified on the DTAMBRS parameter in the same order in which they are specified in DDS. For example:

```
DTAMBRS((PF1 M1) (PF2(M1 M2)) (PF1 M1) (PF2(M1 M2)) (PF3 *NONE))
```

The same physical file name can be specified for more than one PFILE keyword. In this case, each occurrence of the file name is handled as a different physical file and should be specified as many times as it is specified on the DTAMBRS parameter.

If a library name is not specified for the file on the PFILE keyword, the library list is used to find the physical file when the logical file is created. The physical file name and the library name then become part of the logical file description. The physical file names and the library names specified on the DTAMBRS parameter must be the same as those stored in the logical file description. If a file name is not qualified by a library name on the DTAMBRS parameter, the library name defaults to *CURRENT, and the system uses the library name that is stored in the logical file description for the respective physical file name. This library name is either the library name that was specified for the file on the PFILE DDS keyword or the name of the library in which the file was found using the library list. When you add a member to a logical file, you can specify data members as follows:

- Specify no associated physical file members (DTAMBRS parameter). The member is associated with all the physical file members of all physical files in all PFILE keywords specified in the logical file DDS.
- Specify the associated physical file members (DTAMBRS parameter). If you do not specify library names, the logical file determines the libraries used. If more than one physical file with the same name is specified, you must specify the library names. When more than one physical file member is specified for a physical file, the member names should be specified in the order in which records are to be retrieved when duplicate key values occur across those members. If you do not want to include any members from a particular physical file, specify the physical file name and *NONE for the member name (DTAMBRS parameter). No members from the physical file specified by *NONE are included. This method can be used to define a logical file member that contains a subset of the record formats defined for the logical file.

When you define a record format for a logical file that shares the key specifications of another file's access path (using the DDS keyword, REFACCPTH), you can use any fields from the associated physical file record format. The fields do not have to be used in the file that describes the access path. However, all key and select/omit fields used in the file that describes the access path must be used in the new record format.

You can use the Create Logical File (CRTLF) command to create the first member when you create the logical file. Subsequent members must be added using the Add Logical File Member (ADDLFM) command. However, if you are going to add more members, you must specify more than 1 for the MAXMBRS parameter. The following example of adding a member to a logical file uses the CRTLF command used earlier in "Creating a Logical File" on page 6-13.

```
CRTLF   FILE(DSTPRODLB/ORDHDRL)
        MBR(*FILE)  DTAMBRS(*ALL)
        TEXT('Order header logical file')
```

*FILE is the default for the MBR parameter and means that the name of the member is the same as the name of the file. All the members of the associated physical file (ORDHDRP) are used in the logical file (ORDHDRL) in the logical file member. The text description of the file is also the text description of the member.

Join Logical File Considerations

This section covers the following topics:

- Basic concepts of joining two physical files (Example 1)
- Setting up a join logical file
- Using more than one field to join files (Example 2)
- Handling duplicate records in secondary files using the JDUPSEQ keyword (Example 3)
- Handling join fields whose attributes do not match (Example 4)
- Using fields that never appear in the record format to join files – neither fields (Example 5)
- Specifying key fields in join logical files (Example 6)
- Specifying select/omit statements in join logical files
- Joining three or more physical files (Example 7)
- Joining a physical file to itself (Example 8)
- Using default data for records missing from secondary files – the JDFTVAL keyword (Example 9)
- Describing a complex join logical file (Example 10)
- Performance considerations
- Data integrity considerations
- Summary of rules for join logical files

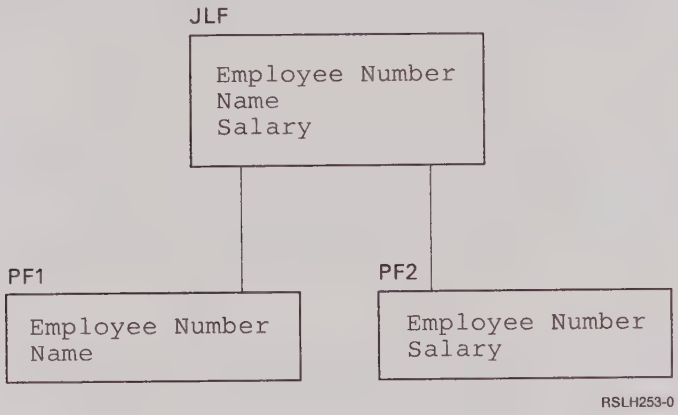
In general, the examples in this section include a picture of the files, DDS for the files, and sample data. For Example 1, several cases are given that show how to join files in different situations (when data in the physical files varies).

In the examples, for convenience and ease of recognition, join logical files are shown with the label JLF, and physical files are illustrated with the labels PF1, PF2, PF3, and so forth.

Basic Concepts of Joining Two Physical Files (Example 1)

A **join logical file** is a logical file that combines (in one record format) fields from two or more physical files. In the record format, not all the fields need to exist in all the physical files.

The following example illustrates a join logical file that joins two physical files. This example is used for the five cases that are discussed in Example 1.



In this example, employee number is common to both physical files (PF1 and PF2), but name is found only in PF1, and salary is found only in PF2.

With a join logical file, the application program does one read operation (to the record format in the join logical file) and gets all the data needed from both physical files. Without the join specification, the logical file would contain two record formats, one based on PF1 and the other based on PF2, and the application program would have to do two read operations to get all the needed data from the two physical files. Thus, join provides more flexibility in designing your data base.

There are, however, a few restrictions on join logical files:

- You cannot change a physical file through a join logical file. To do update, delete, or write (add) operations, you must create a second multiple format logical file and use it to change the physical files. You can also use the physical files, directly, to do the change operations.
- You cannot use DFU to display a join logical file.
- You can specify only one record format in a join logical file.
- The record format in a join logical file cannot be shared.
- A join logical file cannot share the record format of another file.
- Key fields must be fields defined in the join record format and must be fields from the first file specified on the JFILE keyword (which is called the primary file).
- Select/omit fields must be fields defined in the join record format, but can come from any of the physical files.
- Commitment control cannot be used with join logical files.

The following shows the DDS for Example 1:

JLF

```
A                R JOINREC
A                J
A                  JFILE(PF1 PF2)
A                  JOIN(PF1 PF2)
A                  JFLD(NBR NBR)
A                  JREF(PF1)
A                  NBR
A                  NAME
A                  SALARY
A                K NBR
```

PF1

```
A                R REC1
A                  NBR                10
A                  NAME               20
A                K NBR
```

PF2

```
A                R REC2
A                  NBR                10
A                  SALARY              7  2
A                K NBR
```

RSLH254-0

Figure 6-5. DDS Example For Joining Two Physical Files

The following describes the DDS for the join logical file in Example 1 (see the *DDS Reference* for more information on the specific keywords):

The record level specification identifies the record format name used in the join file.

R — Identifies the record format. There can be only one record format in a join logical file.

JFILE — Replaces the PFILE keyword used in simple and multiple format logical files. You must specify at least two physical files. The first file specified on the JFILE keyword is the **primary file**. The other files specified on the JFILE keyword are **secondary files**.

The join specification describes the way a pair of physical files is joined. The second file of the pair is always a secondary file, and there must be one join specification for each secondary file.

J — Identifies the start of a join specification. You must specify at least one join specification in a join logical file. A join specification ends at the first field name specified in positions 19 through 28 or at the next J specified in position 17.

JOIN — Identifies which two files are joined by the join specification. If only two physical files are joined by the join logical file, the JOIN keyword is optional. See "Joining Three or More Physical Files (Example 7)" on page 6-39 later in this section for an example of how to use this keyword.

JFLD — Identifies the **join fields** that join records from the physical files specified on the JOIN keyword. JFLD must be specified at least once for each join specification. The join fields are fields common to the physical files. The first join field is a field from the first file specified on the JOIN keyword, and the second join field is a field from the second file specified on the JOIN keyword. The join fields must have the same attributes (data type, length, and decimal positions). If you are joining physical file fields that do not have the same attributes, you can redefine them for use in a join logical file. See "Using Join Fields Whose Attributes Are Different (Example 4)" on page 6-34 for a description and example.

The field level specification identifies the fields included in the join file.

Field names — Specifies which fields (in this example, *Nbr*, *Name*, and *Salary*) are used by the application program. At least one field name is required. You can specify any field names from the physical files used by the logical file. You can also use keywords like RENAME, CONCAT, or SST as you would in simple and multiple format logical files.

JREF — In the record format (which follows the join specification level and precedes the key field level, if any), the field names must uniquely identify which physical file the field comes from. In this example, the *Nbr* field occurs in both PF1 and PF2. Therefore, the JREF keyword is required to identify the file from which the *Nbr* field description will be used.

The key field level specification is optional, and includes the key field names for the join logical file.

Key field names — Key field names (in this example, *Nbr* is the only key field) are optional and make the join logical file an indexed (keyed sequence) file. Without key fields, the join logical file is an arrival sequence file. In join logical files, key fields must be fields from the primary file, and the key field name must be specified in positions 19 through 28 in the logical file record format.

The select/omit field level specification is optional, and includes select/omit field names for the join logical file.

S or O — Identifies a select or omit specification. The S or O appears in position 17. Select/omit specifications are optional.

Select/omit field names — Only those records meeting the select/omit values will be returned to the program using the logical file. Select/omit fields must be specified in positions 19 through 28 in the logical file record format.

Reading a Join Logical File

The following cases describe how the join logical file in Figure 6-5 presents records to an application program.

The PF1 file is specified first on the JFILE keyword, and is therefore the primary file. When the application program requests a record, the system does the following:

1. Uses the value of the first join field in the primary file (the *Nbr* field in PF1).
2. Finds the first record in the secondary file with a matching join field (the *Nbr* field in PF2 matches the *Nbr* field in PF1).

3. For each match, joins the fields from the physical files into one record and provides this record to your program. Depending on how many records are in the physical files, one of the following conditions could occur:
   a. For all records in the primary file, there is only one matching record in the secondary file. The resulting join logical file will contain a single record for each record in the primary file. See "Matching Records in Primary and Secondary Files (Case 1)" on page 6-26.
   b. For some records in the primary file, there is no matching record in the secondary file.

      If you specify the JDFTVAL keyword:
      - For those records in the primary file that have a matching record in the secondary file, the system will join to the secondary, or multiple secondaries.
      - For those records in the primary file that do not have a matching record in the secondary file, the system will add the default value fields for the secondary file and continue the join operation. You can use the DFT keyword in the physical file to define which defaults are used. See "Record Missing in Secondary File; JDFTVAL Keyword Not Specified (Case 2A)" on page 6-26 and "Record Missing in Secondary File; JDFTVAL Keyword Specified (Case 2B)" on page 6-27.

        Note: If the DFT keyword is specified in the secondary file, the value specified for the DFT keyword is used in the join. The result would be at least one join record for each primary record.

      - If a record exists in the secondary file, but the primary file has no matching value, no record is returned to your program. A second join logical file may be used that reverses the order of primary and secondary files to determine if secondary file records exist with no matching primary file records.

      If you do not specify the JDFTVAL keyword:
      - If a matching record in a secondary file exists, the system will join to the secondary, or multiple secondaries. The result is one or more records for each record in the primary file.
      - If a matching record in a secondary file does not exist, the system will not return a record.

      Note: When the JDFTVAL is not specified, the system will return a record only if there is a match in every secondary file for a record in the primary file.

      In the following examples, cases 1 through 4 describe sequential read operations, and case 5 describes reading by key.

Matching Records in Primary and Secondary Files (Case 1)

Assume that a join logical file is specified as in Figure 6-5 on page 6-23, and that there are four records in both PF1 and PF2, as follows:

PF1

235	Anne
440	Doug
500	Mark
729	Sue

PF2

235	1700.00
440	950.50
500	2100.00
729	1400.90

RSLH255-0

The program does four read operations and gets the following records:

JLF

235	Anne	1700.00
440	Doug	950.50
500	Mark	2100.00
729	Sue	1400.90

RSLH256-0

Record Missing in Secondary File; JDFTVAL Keyword Not Specified (Case 2A)

Assume that a join logical file is specified as in Figure 6-5 on page 6-23, and that there are four records in PF1 and three records in PF2, as follows:

PF1

235	Anne
440	Doug
500	Mark
729	Sue

PF2

235	1700.00
440	950.50
729	1400.90

No record was found for number 500 in PF2.

RSLH257-1

With the join logical file shown in Example 1, the program reads the join logical file and gets the following records:

JLF

235	Anne	1700.00
440	Doug	950.50
729	Sue	1400.90

RSLH258-0

If you do not specify the JDFTVAL keyword and there is no match for the key field in the secondary file, the record is not included in the join logical file.

Record Missing in Secondary File; JDFTVAL Keyword Specified (Case 2B)

Assume that a join logical file is specified as in Figure 6-5 on page 6-23, except that the JDFTVAL keyword is specified, as shown in the following DDS:

JLF

```
     A                                      JDFTVAL
     A                R  JOINREC            JFILE(PF1 PF2)
     A                J                     JOIN(PF1 PF2)
     A                                      JFLD(NBR NBR)
     A                   NBR                JREF(PF1)
     A                   NAME
     A                   SALARY
     A                K  NBR
     A
     A
```

RSLH259-0

The program reads the join logical file and gets the following records:

JLF

```
235     Anne     1700.00
440     Doug      950.50
500     Mark     0000.00   <-- A record for number 500 is returned if JDFTVAL is specified (but the SALARY is 0).
729     Sue      1400.90
```

RSLH260-0

With JDFTVAL specified, the system returns a record for 500, even though the record is missing in PF2. Without that record, some field values may be missing in the join record (in this case, the *Salary* field is missing). With JDFTVAL specified, missing character fields normally default to blanks; missing numeric fields default to zero. However, if the DFT keyword is specified for the field in the physical file, then the default value specified on the DFT keyword is used.

Secondary File Has More Than One Match for a Record in the Primary File (Case 3)

Assume that a join logical file is specified as in Figure 6-5 on page 6-23, and that there are four records in PF1 and five records in PF2, as follows:

PF1

```
235     Anne
440     Doug
500     Mark
729     Sue
```

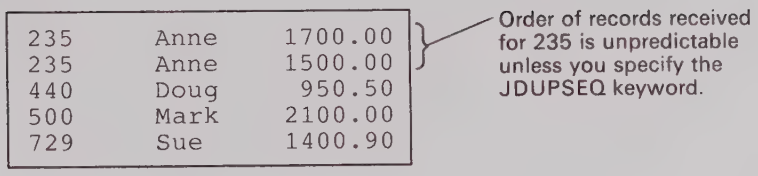
PF2

```
235     1700.00
235     1500.00   <-- Duplicate record was found in PF2 for number 235.
440      950.50
500     2100.00
729     1400.90
```

RSLH261-1

The program gets five records:

JLF



RSLH262-0

For more information, see "Reading Duplicate Records in Secondary Files (Example 3)" on page 6-33.

Extra Record in Secondary File (Case 4)

Assume that a join logical file is specified as in Figure 6-5 on page 6-23, and that there are four records in PF1 and five records in PF2, as follows:

PF1

235	Anne
440	Doug
500	Mark
729	Sue

PF2

235	1700.00
301	1500.00
440	950.50
500	2100.00
729	1400.90

Record for number 301 is only in PF2.

RSLH263-1

The program reads the join logical file and gets only four records, which would be the same even if JDFTVAL were specified (because there must always be a record in the primary file to get a join record):

JLF

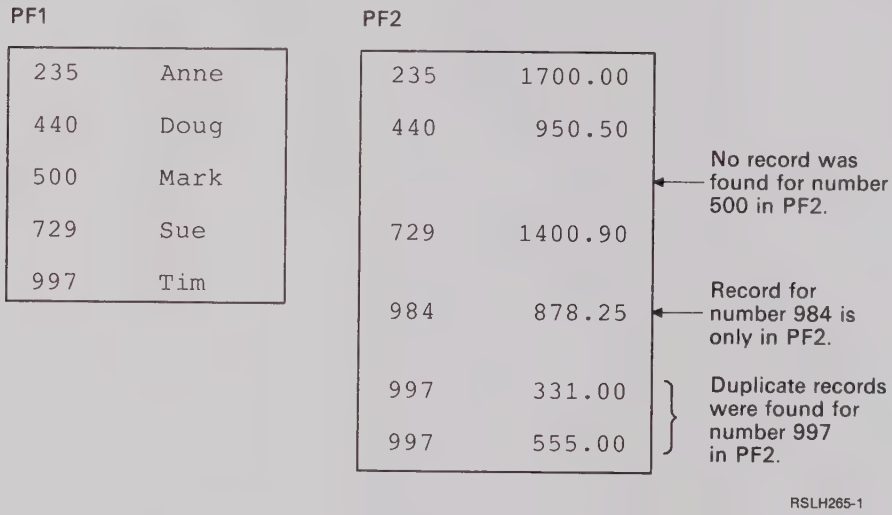
235	Anne	1700.00
440	Doug	950.50
500	Mark	2100.00
729	Sue	1400.90

RSLH264-0

Random Access (Case 5)

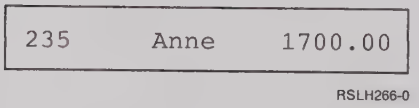
Assume that a join logical file is specified as in Figure 6-5 on page 6-23, Note that the join logical file has key fields defined. This case shows which records would be returned for a random access read operation using the join logical file.

Assume that PF1 and PF2 have the following records:

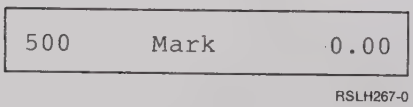


The program can get the following records:

Given a value of 235 from the program for the *Nbr* field in the logical file, the system supplies the following record:



Given a value of 500 from the program for the *Nbr* field in the logical file and with the JDFTVAL keyword specified, the system supplies the following record:



Note: If the JDFTVAL keyword were not specified in the join logical file, no record would be found for a value of 500 because there is no matching record in the secondary file.

Given a value of 984 from the program for the *Nbr* field in the logical file, the system supplies no record and a no record found exception occurs because record 984 is not in the primary file.

Given a value of 997 from the program for the *Nbr* field in the logical file, the system returns one of the following records:

```
997      Tim      331.00
```

or

```
997      Tim      555.00
```

RSLH268-0

Which record is returned to the program cannot be predicted. To specify which record is returned, specify the JDUPSEQ keyword in the join logical file. See "Reading Duplicate Records in Secondary Files (Example 3)" on page 6-33.

Notes:

1. With random access, the application programmer must be aware that there could be duplicate records in PF2, and ensure that the program does more than one read operation for records with duplicate keys. If the program were using sequential access, a second read operation would get the second record.
2. If you specify the JDUPSEQ keyword, the system may create a separate access path for the join logical file (because there is less of a chance the system will find an existing access path that it can share). If you omit the JDUPSEQ keyword, the system may be able to share the access path of another file. (In this case, the system would share the access path of PF2.)

Setting Up a Join Logical File

To set up a join logical file, do the following:

1. Find the field names of all the physical file fields you want in the logical file record format. (You can display the fields contained in files by using the Display File Field Descriptions [DSPFFD] command.)
2. Describe the fields in the record format. Write the field names in a vertical list. This is the start of the record format for the join logical file.

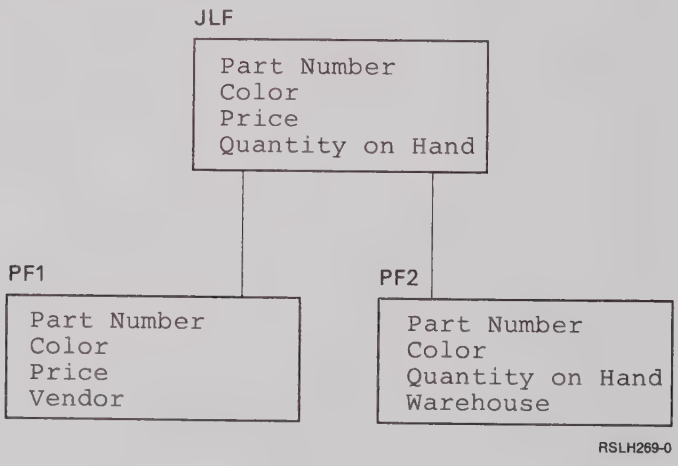
   Note: You can specify the field names in any order. If the same field names appear in different physical files, specify the name of the physical file on the JREF keyword for those fields. You can rename fields using the RENAME keyword, and concatenate fields from the same physical file using the CONCAT keyword. You can also specify a character string which is a subset of an existing character field or zoned field using the SST keyword. You can redefine fields, changing their data type, length, or decimal positions.

3. Specify the names of the physical files as parameter values on the JFILE keyword. The first name you specify is the primary file. The others are all secondary files. For best performance, specify the secondary files with the least records first after the primary file.
4. For each secondary file, code a join specification. On each join specification, identify which pair of files are joined (using the JOIN keyword; optional if only one secondary file), and identify which fields are used to join the pair (using the JFLD keyword; at least one required in each join specification).

5. Optionally, specify the following:
   a. The JDFTVAL keyword. Do this if you want to return a record for each record in the primary file even if no matching record exists in a secondary file.
   b. The JDUPSEQ keyword. Do this for fields that might have duplicate values in the secondary files. JDUPSEQ specifies on which field (other than one of the join fields) to sort these duplicates, and the sequence that should be used.
   c. Key fields. Key fields cannot come from a secondary file. If you omit key fields, records are returned in arrival sequence as they appear in the primary file.
   d. Select/omit fields. In some situations, you must also specify the dynamic selection (DYNSLT) keyword at the file level. For details, see the *DDS Reference*.
   e. Neither fields. For a description, see "Describing Fields That Never Appear in the Record Format (Example 5)" on page 6-36.

Using More Than One Field to Join Files (Example 2)

You can specify more than one join field to join a pair of files. The following shows the fields in the logical file and the two physical files.



The DDS for these files is as follows:

JLF

```
A                R JOINREC
A                J
A
A
A                  PTNBR
A                  COLOR
A                  PRICE
A                  QUANTOH
A
A
```

```
A                                  JFILE(PF1 PF2)
A                                  JOIN(PF1 PF2)
A                                  JFLD(PTNBR PTNBR)
A                                  JFLD(COLOR COLOR)
A                                  JREF(PF1)
A                                  JREF(PF1)
```

PF1

```
A                R REC1
A                  PTNBR          4
A                  COLOR         20
A                  PRICE          7  2
A                  VENDOR        40
A
```

PF2

```
A                R REC2
A                  PTNBR          4
A                  COLOR         20
A                  QUANTOH        5  0
A                  WAREHSE       30
A
```

RSLH270-1

Assume that the physical files have the following records:

PF1

100	Black	22.50	ABC Corp.
100	White	20.00	Ajax Inc.
120	Yellow	3.75	ABC Corp.
187	Green	110.95	ABC Corp.
187	Red	110.50	ABC Corp.
190	Blue	40.00	Ajax Inc.

PF2

100	Black	23	Boston
100	White	15	Boston
120	Yellow	102	Concord
187	Green	0	Boston
187	Red	2	Concord
190	White	2	Concord

RSLH271-0

If the file is processed sequentially, the program receives the following records:

JLF

100	Black	22.50	23
100	White	20.00	15
120	Yellow	3.75	102
187	Green	110.95	0
187	Red	110.50	2

RSLH272-0

Note that no record for part number 190, color blue, is available to the program, because there is not a match on both fields in the secondary file. Because JDFTVAL was not specified, no record is returned.

Reading Duplicate Records in Secondary Files (Example 3)

Sometimes a join to a secondary file produces more than one record from the secondary file. When this occurs, specifying the JDUPSEQ keyword in the join specification for that secondary file tells the system to base the order of the duplicate records on the specified field in the secondary file. The DDS for the physical files and for the join logical file are as follows:

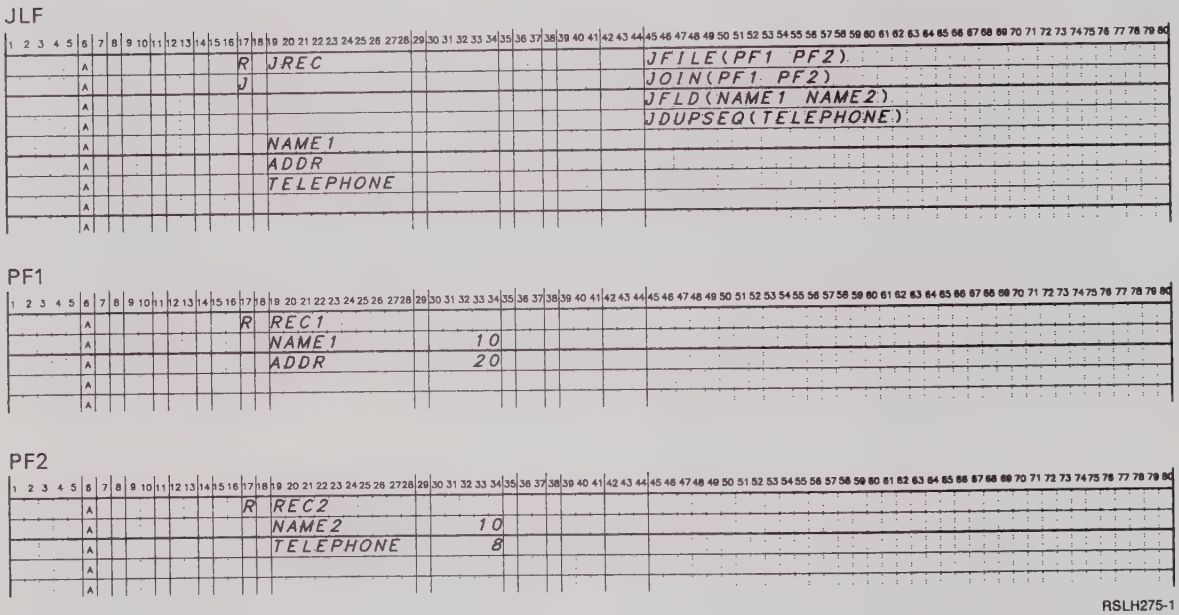


Figure 6-6. DDS Example Using the JDUPSEQ Keyword

The physical files have the following records:

PF1

```
Anne    120 1st St.
Doug    40 Pillsbury
Mark    2 Lakeside Dr.
```

PF2

```
Anne    555-1111
Anne    555-6666
Anne    555-2222
Doug    555-5555
```

RSLH273-0

The join logical file returns the following records:

JLF

Anne	120 1st St.	555-1111
Anne	120 1st St.	555-2222
Anne	120 1st St.	555-6666
Doug	40 Pillsbury	555-5555

Anne's telephone numbers are in ascending order.

RSLH274-1

The program reads all the records available for Anne, then Doug, then Mark. Anne has one address, but three telephone numbers. Therefore, there are three records returned for Anne.

The records for Anne sort in ascending sequence by telephone number because the JDUPSEQ keyword sorts in ascending sequence unless you specify *DESCEND as the keyword parameter. The following example shows the use of *DESCEND in DDS:

JLF

```
A                     R JREC
A                     J
A                       JFILE(PF1 PF2)
A                       JOIN(PF1 PF2)
A                       JFLD(NAME1 NAME2)
A                       JDUPSEQ(TELEPHONE *DESCEND)
A                     NAME1
A                     ADDR
A                     TELEPHONE
A
A
```

RSLH276-1

When you specify JDUPSEQ with *DESCEND, the records are returned as follows:

JLF

Anne	120 1st St.	555-6666
Anne	120 1st St.	555-2222
Anne	120 1st St.	555-1111
Doug	40 Pillsbury	555-5555

Anne's telephone numbers are in descending order.

RSLH277-1

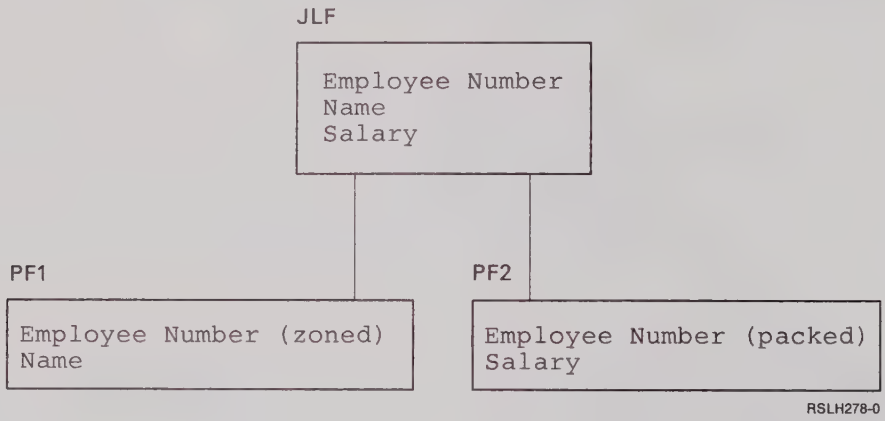
Note: The JDUPSEQ keyword applies only to the join specification in which it is specified. For an example showing the JDUPSEQ keyword in a join logical file with more than one join specification, see "A Complex Join Logical File (Example 10)" on page 6-45.

Using Join Fields Whose Attributes Are Different (Example 4)

Fields from physical files that you are using as join fields generally have the same attributes (length, data type, and decimal positions). For example, as in Figure 6-6, the *Name1* field is a character field 10 characters long in physical file PF1, and can be joined to the *Name2* field, a character field 10 characters long in physical file PF2. The *Name1* and *Name2* fields have the same characteristics and, therefore, can easily be used as join fields.

The following is an example in which the join fields do not have the same attributes. The *Nbr* field in physical file PF1 and the *Nbr* field in physical file PF2 both have a length of 3 specified in position 34, but in the PF1 file the field is zoned (S in position 35), and in the PF2 file the field is packed (P in position 35). To join the two files using these fields as join fields, you must redefine one or both fields to have the same attributes.

The following illustrates the fields in the logical and physical files:



RSLH278-0

The DDS for these files is as follows:

JLF

```
A                R JOINREC
A                J
A                                                  JFILE(PF1 PF2)
A                                                  JOIN(PF1 PF2)
A                                                  JFLD(NBR NBR)
A                  NBR                        S    JREF(2)
A                  NAME
A                  SALARY
```

PF1

```
A                R REC1
A                  NBR                      3S 0  <-- Zoned
A                  NAME                     20
A                K NBR
```

PF2

```
A                R REC2
A                  NBR                      3P 0  <-- Packed
A                  SALARY                    7 2
A                K NBR
```

RSLH279-0

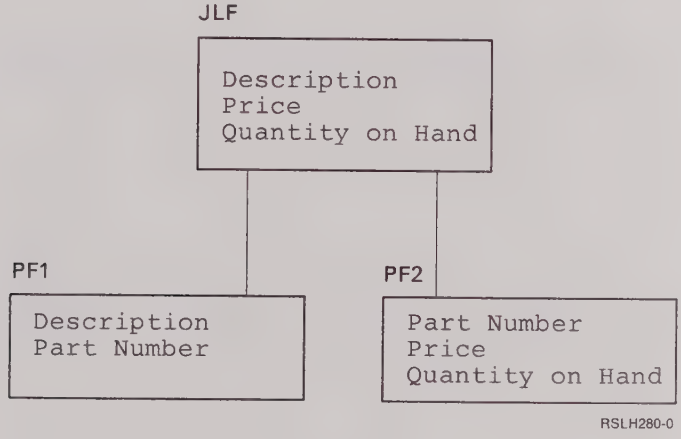
Note: In this example, the *Nbr* field in the logical file comes from PF2, because JREF(2) is specified. Instead of specifying the physical file name, you can specify a relative file number on the JREF keyword; in this example, the 2 indicates PF2.

Because the *Nbr* fields in the PF1 and PF2 files are used as the join fields they must have the same attributes. In this example, they don't. Therefore, you must redefine one or both of them to have the same attributes. In this example, to resolve the difference in the attributes of the two employee number fields, the *Nbr* field in JLF (which is coming from the PF2 file) is redefined as zoned (S in position 35 of JLF).

Describing Fields That Never Appear in the Record Format (Example 5)

A neither field (N specified in position 38) can be used in join logical files for neither input nor output. Programs using the join logical file cannot see or read neither fields. Neither fields are not included in the record format. Neither fields cannot be key fields or used in select/omit statements in the joined file. You can use a neither field for a join field (specified at the join specification level on the JFLD keyword) that is redefined at the record level only to allow the join, but is not needed or wanted in the program.

In the following example, the program reads the descriptions, prices, and quantity on hand of parts in stock. The part numbers themselves are not wanted except to bring together the records of the parts. However, because the part numbers have different attributes, at least one must be redefined.



The DDS for these files is as follows:

JLF

```
A                R JOINREC
A                J
A                  JFILE(PF1 PF2)
A                  JOIN(PF1 PF2)
A                  JFLD(PRTNBR PRTNBR)
A                  PRTNBR                  S   N
A                  JREF(1)
A                  DESC
A                  PRICE
A                  QUANT
A                K DESC
```

PF1

```
A                R REC1
A                  DESC                30
A                  PRTNBR               6P  0
```

PF2

```
A                R REC2
A                  PRTNBR               6S  0
A                  PRICE                7   2
A                  QUANT                8   0
```

RSLH281-0

In PF1, the *Prtnbr* field is a packed decimal field; in PF2, the *Prtnbr* field is a zoned decimal field. In the join logical file, they are used as join fields, and the *Prtnbr* field from PF1 is redefined to be a zoned decimal field by specifying an S in position 35 at the field level. The JREF keyword identifies which physical file the field comes from. However, the field is not included in the record format; therefore, N is specified in position 38 to make it a neither field. A program using this file would not see the field.

In this example, a sales clerk can type a description of a part. The program can read the join logical file for a match or a close match, and display one or more parts for the user to examine, including the description, price, and quantity. This application assumes that part numbers are not necessary to complete a customer order or to order more parts for the warehouse.

Specifying Key Fields in Join Logical Files (Example 6)

If you specify key fields in a join logical file, the following rules apply:

- The key fields must exist in the primary physical file.
- The key fields must be named in the join record format in the logical file in positions 19 through 28.
- The key fields cannot be fields defined as neither fields (N specified in position 38 for the field) in the logical file.

The following illustrates the rules for key fields:

JLF

```
A            R JOINREC                  JFILE(PF1 PF2)
A            J                          JOIN(PF1 PF2)
A                                       JFLD(NBR NUMBER)
A                                       JFLD(FLD3 FLD31)
A              FLD1                     RENAME(F1)
A              FLD2                     JREF(2)
A              FLD3                35 N
A              NAME
A              TELEPHONE                CONCAT(AREA LOCAL)
A            K FLD1
A            K NAME
```

PF1

```
A            R REC1
A              NBR                  4
A              F1                  20
A              FLD2                 7  2
A              FLD3                40
A              NAME                20
```

PF2

```
A            R REC2
A              NUMBER               4
A              FLD2                 7  2
A              FLD31               35
A              AREA                 3
A              LOCAL                7
```

RSLH282-1

The following fields **cannot** be key fields:

Nbr (not named in positions 19 through 28)
Number (not named in positions 19 through 28)
F1 (not named in positions 19 through 28)
Fld31 (comes from a secondary file)
Fld2 (comes from a secondary file)
Fld3 (is a neither field)
Area and *Local* (not named in positions 19 through 28)
Telephone (is based on fields from a secondary file)

Specifying Select/Omit Statements in Join Logical Files

If you specify select/omit statements in a join logical file, the following rules apply:

- The fields can come from any physical file the logical file uses (specified on the JFILE keyword).
- The fields you specify on the select/omit statements cannot be fields defined as neither fields (N specified in position 38 for the field).
- In some circumstances, you must specify the DYNSLT keyword when you specify select/omit statements in join logical files. For more information and examples, see the DYNSLT keyword in the *DDS Reference*.

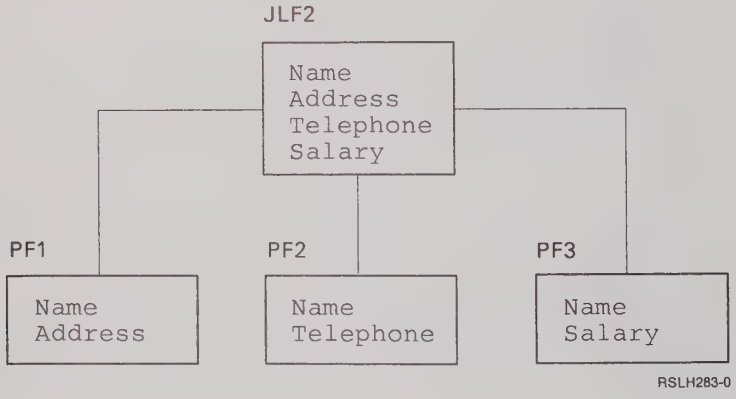
For an example showing select/omit statements in a join logical file, see "A Complex Join Logical File (Example 10)" on page 6-45.

Joining Three or More Physical Files (Example 7)

You can use a join logical file to join as many as 32 physical files. These files must be specified on the JFILE keyword. The first file specified on the JFILE keyword is the primary file; the other files are all secondary files.

The physical files must be joined in pairs, with each pair described by a join specification. Each join specification must have one or more join fields identified.

The following shows the fields in the files and one field common to all the physical files in the logical file:



In this example, the *Name* field is common to all the physical files (PF1, PF2, and PF3), and serves as the join field.

The following shows the DDS for the physical and logical files:

JLF

```
A*
A*  JLF
A*
A          R JOINREC
A            J
A            JFILE(PF1 PF2 PF3)
A            JOIN(PF1 PF2)
A              JFLD(NAME NAME)
A            J
A            JOIN(PF2 PF3)
A              JFLD(NAME NAME)
A              NAME
A              JREF(PF1)
A              ADDR
A              TELEPHONE
A              SALARY
A            K NAME
```

PF1

```
A*
A*  PF1
A*
A          R REC1
A            NAME          10
A            ADDR          20
A            K NAME
```

PF2

```
A*
A*  PF2
A*
A          R REC2
A            NAME          10
A            TELEPHONE      7
A            K NAME
```

PF3

```
A*
A*  PF3
A*
A          R REC3
A            NAME          10
A            SALARY         9  2
A            K NAME
```

RSLH284-1

Assume the physical files have the following records:

PF1

```
Anne    120 1st St.
Doug    40 Pillsbury
Mark    2 Lakeside Dr.
Tom     335 Elm St.
```

PF2

```
Anne    555-1111
Doug    555-5555
Mark    555-0000
Sue     555-3210
```

PF3

```
Anne    1700.00
Doug     950.50
Mark    2100.00
```

RSLH285-0

The program reads the following logical file records:

JLF

```
Anne    120 1st St.       555-1111    1700.00
Doug    40 Pillsbury      555-5555     950.50
Mark    2 Lakeside Dr.    555-0000    2100.00
```

RSLH286-0

No record is returned for Tom because there is no record for him in PF2 and PF3 and the JDFTVAL keyword is not specified. No record is returned for Sue because the primary file has no record for Sue.

Joining a Physical File to Itself (Example 8)

You can join a physical file to itself to read records that are formed by combining two or more records from the physical file itself. The following example shows how:

JLF

```
Employee Number
Name
Manager's Name
```

PF1

```
Employee Number
Name
Manager's Employee Number
```

RSLH287-0

The following shows the DDS for these files:

JLF

```
     A                                      JDFTVAL
     A            R JOINREC                 JFILE(PF1 PF1)
     A            J                         JOIN(1 2)
     A                                      JFLD(MGRNBR NBR)
     A              NBR                     JREF(1)
     A              NAME                    JREF(1)
     A              MGRNAME                 RENAME(NAME)
     A                                      JREF(2)
     A
     A
```

PF1

```
     A            R RCD1
     A              NBR              3
     A              NAME            10      DFT('none')
     A              MGRNBR           3
     A
     A
```

RSLH288-0

Notes:

1. Relative file numbers must be specified on the JOIN keyword because the same file name is specified twice on the JFILE keyword. Relative file number 1 refers to the first physical file specified on the JFILE keyword, 2 refers to the second, and so forth.
2. With the same physical files specified on the JFILE keyword, the JREF keyword is required for each field specified at the field level.

Assume there are the following records in PF1:

PF1

```
235     Anne    440
440     Doug    729
500     Mark    440
729     Sue     888
```

RSLH289-0

The program reads the following logical file records:

JLF

```
235     Anne    Doug
440     Doug    Sue
500     Mark    Doug
729     Sue     none
```

RSLH290-0

Note that a record is returned for the manager name of Sue because the JDFTVAL keyword was specified. Also note that the value none is returned because the DFT keyword was used on the *Name* field in the PF1 physical file.

Using Default Data for Missing Records from Secondary Files (Example 9)

If you are joining more than two files, and you specify the JDFTVAL keyword, the default value supplied by the system for a join field missing from a secondary file is used to join to other secondary files. If the DFT keyword is specified in the secondary file, the value specified for the DFT keyword is used in the logical file.

The DDS for the files is as follows:

JLF

```
     A                                      JDFTVAL
     A                R     JRCD            JFILE(PF1 PF2 PF3)
     A                J                     JOIN(PF1 PF2)
     A                                      JFLD(NAME NAME)
     A                J                     JOIN(PF2 PF3)
     A                                      JFLD(TELEPHONE TELEPHONE)
     A                      NAME            JREF(PF1)
     A                      ADDR
     A                      TELEPHONE       JREF(PF2)
     A                      LOC
     A
     A
```

PF1

```
     A                R     RCD1
     A                      NAME      20
     A                      ADDR      40
     A                      COUNTRY   40
     A
     A
```

PF2

```
     A                R     RCD2
     A                      NAME      20
     A                      TELEPHONE  8          DFT('999-9999')
     A
     A
```

PF3

```
     A                R     RCD3
     A                      TELEPHONE  8
     A                      LOC       30          DFT('No location assigned')
     A
     A
```

RSLH291-1

Assume that PF1, PF2, and PF3 have the following records:

PF1

```
Anne    120 1st St.       USA
Doug    40 Pillsbury      Canada
Mark    2 Lakeside Dr.    Canada
Sue     120 Broadway      USA
```

PF2

```
Anne     555-1234
Doug     555-2222
Sue      555-1144
```

← No telephone number was found for Mark in PF2.

PF3

```
555-1234     Room 312
555-2222     Main lobby
999-9999     No telephone number
```

← No record for telephone number 555-1144 was found in PF3.

RSLH292-1

With JDFTVAL specified in the join logical file, the program reads the following logical file records:

```
Anne     120 1st St.        555-1234     Room 312
Doug     40 Pillsbury       555-2222     Main lobby
Mark     2 Lakeside Dr.     999-9999     No telephone number
Sue      120 Broadway       555-1144     No location assigned
```

RSLH293-0

In this example, there is complete data for Anne and Doug. However, part of the data is missing for Mark and Sue.

- PF2 is missing a record for Mark because he has no telephone number. The default value for the *Telephone* field in PF2 is defined as 999-9999 using the DFT keyword. In this example, therefore, 999-9999 is the telephone number returned when no telephone number is assigned. The JDFTVAL keyword specified in the join logical file causes the default value for the *Telephone* field (which is 999-9999) in PF2 to be used to match with a record in PF3. (In PF3, a record is included to show a description for telephone number 999-9999.) Without the JDFTVAL keyword, no record would be returned for Mark.
- Sue's telephone number is not yet assigned a location; therefore, a record for 555-1144 is missing in PF3. Without JDFTVAL specified, no record would be returned for Sue. With JDFTVAL specified, the system supplies the default value

specified on the DFT keyword in PF3 the *Loc* field (which is `No location assigned`).

A Complex Join Logical File (Example 10)

The following example shows a more complex join logical file. Assume the data is in the following three physical files:

Vendor Master File (PF1)

```
A                                      R RCD1                          TEXT('VENDOR INFORMATION')
A                                        VDRNBR                  5     TEXT('VENDOR NUMBER')
A                                        VDRNAM                 25     TEXT('VENDOR NAME')
A                                        STREET                 15     TEXT('STREET ADDRESS')
A                                        CITY                   15     TEXT('CITY')
A                                        STATE                   2     TEXT('STATE')
A                                        ZIPCODE                 5     TEXT('ZIP CODE')
A                                                                      DFT('00000')
A                                        PAY                     1     TEXT('PAY TERMS')
A
A
```

Order File (PF2)

```
A                                      R RCD2                          TEXT('VENDORS ORDER')
A                                        VDRNUM                 5S 0   TEXT('VENDOR NUMBER')
A                                        JOBNBR                 6      TEXT('JOB NUMBER')
A                                        PRTNBR                 5S 0   TEXT('PART NUMBER')
A                                                                      DFT(99999)
A                                        QORDER                 3S 0   TEXT('QUANTITY ORDERED')
A                                        UNTPRC                 6S 2   TEXT('PRICE')
A
A
```

Part File (PF3)

```
A                                      R RCD3                          TEXT('DESCRIPTION OF PARTS')
A                                        PRTNBR                 5S 0   TEXT('PART NUMBER')
A                                                                      DFT(99999)
A                                        DESCR                  25     TEXT('DESCRIPTION')
A                                        UNITPRICE              6S 2   TEXT('UNIT PRICE')
A                                        WHSNBR                 3      TEXT('WAREHOUSE NUMBER')
A                                        PRTLOC                 4      TEXT('LOCATION OF PART')
A                                        QOHAND                 5      TEXT('QUANTITY ON HAND')
A
A
```

RSLH294-0

The join logical file record format should contain the following fields:

Vdrnam (vendor name)
Street, *City*, *State*, and *Zipcode* (vendor address)
Jobnbr (job number)
Prtnbr (part number)
Descr (description of part)
Qorder (quantity ordered)
Untprc (unit price)
Whsnbr (warehouse number)
Prtloc (location of part)

The DDS for this join logical file is as follows:

Join Logical File (JLF)

```
1 DYNSLT
2 JDFTVAL
          R RECORD1                    JFILE(PF1 PF2 PF3)
3         J                            JOIN(1 2)
                                       JFLD(VDRNBR VDRNUM)
4                                      JDUPSEQ(JOBNBR)
5         J                            JOIN(2 3)
                                       JFLD(PRTNBR PRTNBR)
6                                      JFLD(UNTPRC UNITPRICE)
7           VDRNUM          5A   N     TEXT('CHANGED ZONED TO CHARACTER')
            VDRNAM
            ADDRESS                    CONCAT(STREET CITY STATE +
8                                        ZIPCODE)
            JOBNBR
            PRTNBR                     JREF(2)
9
            DESCR
            QORDER
            UNTPRC
            WHSNBR
            PRTLOC
10          S VDRNAM                   COMP(EQ 'SEWING COMPANY')
            S QORDER                   COMP(GT 5)
```

RSLH295-0

1. The DYNSLT keyword is required because the JDFTVAL keyword and select fields are specified.
2. The JDFTVAL keyword is specified to pick up default values in physical files.
3. First join specification.
4. The JDUPSEQ keyword is specified because duplicate vendor numbers occur in PF2.
5. Second join specification.
6. Two JFLD keywords are specified to ensure the correct records are joined from the PF2 and PF3 files.
7. The *Vdrnum* field is redefined from zoned decimal to character (because it is used as a join field and it does not have the same attributes in PF1 and PF2).
8. The CONCAT keyword concatenates four fields from the same physical file into one field.
9. The JREF keyword must be specified because the *Prtnbr* field exists in two physical files and you want to use the one in PF2.
10. The select/omit fields are *Vdrnam* and *Qorder*. (Note that they come from two different physical files.)

Performance Considerations

You can do the following to improve the performance of join logical files:

- If the physical files you are joining have a different number of records, specify the physical file with fewest records first (first parameter following the JOIN keyword).
- Consider using the DYNSLT keyword. See "Dynamic Select/Omit" on page 6-11 for more details.
- Consider describing your join logical file so it can automatically share an existing access path. See "Using Existing Access Paths" on page 6-12 for more details.

Note: Join logical files always have access paths using the second field of the pair of fields specified in the JFLD keyword. This field acts like a key

field in simple logical files. If an access path does not already exist, the access path is implicitly created with immediate maintenance.

Data Integrity Considerations

Unless you have a lock on the physical files used by the join logical file, the following can occur:

- Your program reads a record for which there are two or more records in a secondary file. The system supplies one record to your program.
- Another program updates the record in the primary file that your program has just read, changing the join field.
- Your program issues another read request. The system supplies the next record based on the current (new) value of the join field in the primary file.

These same considerations apply to secondary files as well.

Summary of Rules for Join Logical Files

Requirements

The principal requirements for join logical files are:

- Each join logical file must have:
  - Only one record format, with the JFILE keyword specified for it.
  - At least two physical file names specified on the JFILE keyword. (The physical file names on the JFILE keyword do not have to be different files.)
  - At least one join specification (J in position 17 with the JFLD keyword specified).
  - A maximum of 31 secondary files.
  - At least one field name with field use other than N (neither) at the field level.
- If only two physical files are specified for the JFILE keyword, the JOIN keyword is not required. There can be only one join specification, and it joins the two physical files.
- If more than two physical files are specified for the JFILE keyword, the following rules apply:
  - The primary file must be the first file of the pair of files specified on the first JOIN keyword (the primary file can also be the first of the pair of files specified on other JOIN keywords).

    Note: Relative file numbers must be specified on the JOIN keyword when the same file name is specified twice on the JFILE keyword.
  - Every secondary file must be specified only once as the second file of the pair of files on the JOIN keyword. This means that for every secondary file on the JFILE keyword, there must be one join specification (two secondary files would mean two join specifications, three secondary files would mean three join specifications).
  - The order in which secondary files appear in join specifications must match the order in which they are specified on the JFILE keyword.

Join Fields

The rules to remember about join fields are:

- Every physical file you are joining must be joined to another physical file by at least one join field. A join field is a field specified as a parameter value on the JFLD keyword in a join specification.
- Join fields (specified on the JFLD keyword) must have identical attributes (length, data type, and decimal positions) or be redefined in the record format of the join logical file to have the same attributes.
- Join fields need not be specified in the record format of the join logical file (unless you must redefine one or both so that their attributes are identical).
- If you redefine a join field, you can specify N in position 38 (making it a neither field) to prevent a program using the join logical file from using the redefined field.

Fields in Join Logical Files

The rules to remember about fields in join logical files are:

- Fields in a record format for a join logical file must exist in one of the physical files used by the logical file or, if CONCAT, RENAME, TRNTBL, or SST is specified for the field, be a result of fields in one of the physical files.
- Fields specified as parameter values on the CONCAT keyword must be from the same physical file. If the first field name specified on the CONCAT keyword is not unique among the physical files, you must specify the JREF keyword for that field to identify which file contains the field descriptions you want to use.
- If a field name in the record format for a join logical file is specified in more than one of the physical files, you must uniquely specify on the JREF keyword which file the field comes from.
- Key fields, if specified, must come from the primary file. Key fields in the join logical file need not be key fields in the primary file.
- Select/omit fields can come from any physical file used by the logical file, but in some circumstances the DYNSLT keyword is required (see the DYNSLT keyword description in the *DDS Reference*).
- If specified, key fields and select/omit fields must be defined in the record format.
- Relative file numbers must be used for the JOIN and JREF keywords if the name of the physical file is specified more than once on the JFILE keyword.

Miscellaneous

Other rules to keep in mind when using join logical files are:

- Join logical files are read-only files.
- Join record formats cannot be shared, and cannot share other record formats.
- The following are not allowed in a join logical file:
  - The REFACCPTH and FORMAT keywords
  - Both fields (B specified in position 38)

Chapter 7. Data Base Security

This chapter describes some of the data base file security functions. The topics covered include data base file security, public authority considerations, restricting the ability to change or delete any data in a file, and using logical files to secure data. For more information about using the security function on the AS/400 system, see the *Security Concepts and Planning*.

File and Data Authority

The following describes the types of authority that can be granted to a user for a data base file.

Object Operational Authority

Object operational authority is required to:

- Open the file for processing. (You must also have at least one data authority.)
- Compile a program which uses the file description.
- Display descriptive information about active members of a file.
- Open the file for query processing. For example, the Open Query File (OPNQRYF) command opens a file for query processing.

Note: You must also have the appropriate data authorities required by the options specified on the open operation.

Object Existence Authority

Object existence authority is required to:

- Delete the file.
- Save, restore, and free the storage of the file. This authority may also be implicitly granted for all objects by using the *SAVSYS special user authority. (For more information about the *SAVSYS authority, see the *Security Concepts and Planning*.)
- Remove members from the file.
- Transfer ownership of the file.

Note: All these functions except save/restore also require object operational authority to the file.

Object Management Authority

Object management authority is required to:

- Create a logical file with a keyed access path (object management authority is required for the physical file referred to by the logical file).
- Grant and revoke authority. You can grant and revoke only the authority that you already have. (You must also have object operational authority to the file.)
- Change the file.
- Add members to the file. (The owner of the file becomes the owner of the new member.)
- Change the member in the file.
- Move the file.
- Rename the file.
- Rename a member of the file.
- Clear a member of the file. (Delete data authority is also required.)
- Initialize a member of the file. (Add data authority is also required to initialize with default records; delete data authority is required to initialize with deleted records.)
- Reorganize a member of the file. (All data authorities are also required.)

Data Authorities

Data authorities can be granted only to physical files. Logical files use the data authority granted to the physical files associated with the logical file.

Read Authority: You can read the records in the file.

Add Authority: You can add new records to the file.

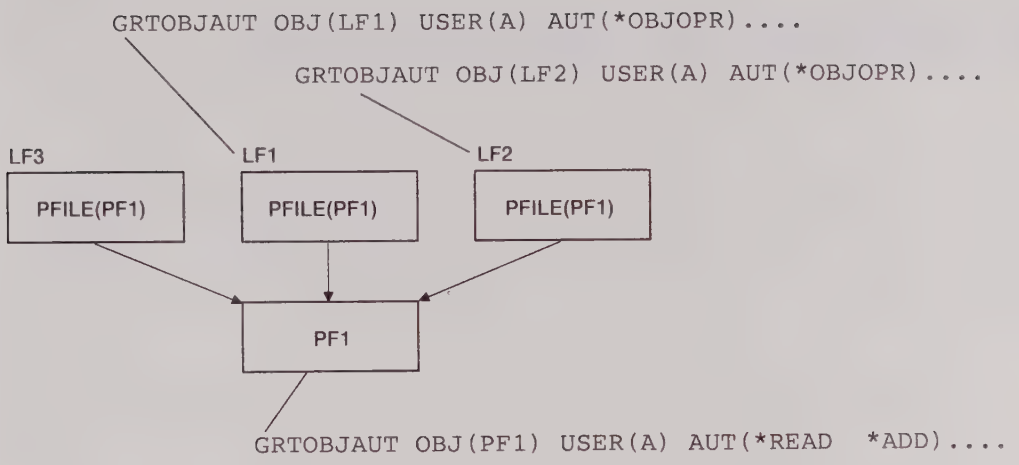
Update Authority: You can update existing records. (To read a record for update, you must also have read authority.)

Delete Authority: You can delete existing records. (To read a record for deletion, you must also have read authority.)

Normally, the authority you have to the data in the file is not verified until you actually perform the input/output operation. However, the Open Query File (OPNQRYF) and Open Data Base File (OPNDBF) commands also verify data authority when the file is opened.

For a physical file, authority to the file and to the data in the file can be granted. For a logical file, authority to the file can be granted, but because logical files contain no data, data authority cannot be granted. If object operational authority is not granted to a user for a physical file, then that user cannot open the physical file.

The following example shows the relationship between authority granted for logical files and the physical files used by the logical file. The logical files LF1, LF2, and LF3 are based on the physical file PF1. USERA has read (*READ) and add (*ADD) authority to the data in PF1 and object operational authority for LF1 and LF2. This means that USERA cannot open PF1 or use its data directly in any way because the user does *not* have object operational authority (*OBJOPR) to PF1; USERA can open LF1 and LF2 and read records from and add records to PF1 through LF1 and LF2. Note that the user was not given authority for LF3 and, therefore, cannot use it.



Public Authority

When you create a file, you can specify public authority through the AUT parameter on the create command. **Public authority** is authority available to any user who does not have specific authority to the file or who is not a member of a group that has specific authority to the file. Public authority is the last authority check made. That is, if the user has specific authority to a file or the user is a member of a group with specific authority, then the public authority is not checked. Public authority can be specified as:

- *CHANGE. All users that do not have specific user or group authority to the file have authority to change data in the file. The *CHANGE value is the default public authority.
- *USE. All users that do not have specific user or group authority to the file have authority to read data in the file.
- *EXCLUDE. Only the owner, security officer, users with specific authority, or users who are members of a group with specific authority can use the file.
- *ALL. All users that do not have specific user or group authority to the file have all data authorities along with object operational, object management, and object existence authorities.
- Authorization list name. An authorization list is a list of users and their authorities. The list allows users and their different authorities to be grouped together. For more information about authorization lists, see the *Security Concepts and Planning*.

Note: When creating a logical file, no data authorities are granted. Consequently, *CHANGE is the same as *USE, and *ALL does not grant any data authority.

You can use the Edit Object Authority (EDTOBJAUT), Grant Object Authority (GRTOBJAUT), or Revoke Object Authority (RVKOBJAUT) commands to grant or revoke the public authority of a file. See the *Security Concepts and Planning* for more information about these commands.

Data Base File Capabilities

File capabilities are used to control which input/output operations are allowed for a data base file independent of data base file authority.

When you create a physical file, you can specify if the file is update-capable and delete-capable by using the ALWUPD and ALWDLT parameters on the Create Physical File (CRTPF) and Create Source Physical File (CRTSRCPF) commands. By creating a file that is not update-capable and not delete-capable, you can effectively enforce an environment where data cannot be changed or deleted from a file once the data is written.

File capabilities cannot be explicitly set for logical files. The file capabilities of a logical file are determined by the file capabilities of the physical files it is based on.

You cannot change file capabilities after the file is created. You must delete the file then recreate it with the desired capability. The Display File Description (DSPFD) command can be used to determine the capability of a file.

Using Logical Files to Secure Data

You can use a logical file to prevent a field in a physical file from being viewed. This is accomplished by describing a logical file record format that does not include fields you do not want the user to see. For more information about this subject, see "Describing Logical File Record Formats" on page 6-1.

You can also use a logical file to prevent one or more fields from being changed in a physical file by specifying, for those fields you want to protect, an I (input only) in position 38 of the *DDS Coding Form*. For more information about this subject, see "Describing Field Usage for Logical Files" on page 6-3.

You can use a logical file to secure records in a physical file based on the contents of one or more fields in that record. To secure records based on the contents of a field, use the select and omit keywords when describing the logical file. For more information about this subject, see "Selecting and Omitting Records Using Logical Files" on page 6-7.

Part 3. Processing Data Base Files in Programs

Chapter 8. Run Time Considerations

Before a file is opened for processing, you should consider how you will use the file in the program and job. A better understanding of the run-time file processing parameters can help you avoid unexpected results. In addition, you might improve the performance of your program.

When a file is opened, the attributes in the data base file description are merged with the parameters in the program. Normally, most of the information the system needs for your program to open and process the file is found in the file attributes and in the application program itself.

Sometimes, however, it is necessary to override the processing parameters found in the file and in the program. For example, if you want to process a member of the file other than the first member, you need a way to tell the system to use the member you want to process. The Override with Data Base File (OVRDBF) command allows you to do this. The OVRDBF command also allows you to specify processing parameters that can improve the performance of your job, but that cannot be specified in the file attributes or in the program. The OVRDBF command parameters take precedence over the file and program attributes. For more details about the Override with Data Base File (OVRDBF) command see the *CL Reference*.

This chapter describes the file processing parameters. The parameter values are determined by the high-level language program, the file attributes, and any open or override commands processed before the high-level language program is called. A summary of these parameters and where you specify them can be found in "Run Time Summary" on page 8-15. For more information about the parameters from the high-level language program, see your high-level language guide. For more information about processing parameters from commands, see the *CL Reference* for the following commands:

- Create Physical File (CRTPF)
- Create Logical File (CRTLF)
- Create Source Physical File (CRTSRCPF)
- Add Physical File Member (ADDPFM)
- Add Logical File Member (ADDLFM)
- Change Physical File (CHGPF)
- Change Physical File Member (CHGPFM)
- Change Logical File (CHGLF)
- Change Logical File Member (CHGLFM)
- Change Source Physical File (CHGSRCPF)
- Override with Data Base File (OVRDBF)
- Open Data Base File (OPNDBF)
- Open Query File (OPNQRYF)
- Close File (CLOF)

File and Member Name

FILE and MBR Parameter. Before you can process data in a data base file, you must identify which file and member you want to use. Normally, you specify the file name and, optionally, the member name in your high-level language program. The system then uses this name when your program requests the file to be opened. To override the file name specified in your program and open a different file, you can use the TOFILE parameter on the Override with Data Base File (OVRDBF) command. If no member name is specified in your program, the first member of the file (as defined by the creation date and time) is processed.

If the member name cannot be specified in the high-level language program (some high-level languages do not allow a member name), or you want a member other than the first member, you can use an Override with Data Base File (OVRDBF) command or an open command (OPNDBF or OPNQRYF) to specify the file and member you want to process (using the FILE and MBR parameters).

To process all the members of a file, use the OVRDBF command with the MBR(*ALL) parameter specified. For example, if FILEX has three members and you want to process all the members, you can specify:

```
OVRDBF   FILE(FILEX)  MBR(*ALL)
```

If you specify MBR(*ALL) on the OVRDBF command, your program will read the members in the order they were created. For each member, your program will read the records in keyed or arrival sequence, depending on whether the file is an arrival sequence or keyed sequence file.

File Processing Options

The following section describes several run-time processing options including identifying what file operations will be used by the program, the starting file position, ignoring the keyed sequence access path, specifying how to handle end of file processing, and identifying the length of the record in the file.

Specifying the Type of Processing

OPTION Parameter. When you use a file in a program, the system needs to know what types of operations you plan to use for that file. For example, the system needs to know if you plan to just read data in the file or if you plan to read and update the data. The valid operation options are: input, output, update, and delete. The system determines the options you are using from information you specify in your high-level language program or from the OPTION parameter on the Open Data Base File (OPNDBF) and Open Query File (OPNQRYF) commands.

The system uses the options to determine which operations are allowed in your program. For example, if you open a file for input only and your program tries an output operation, your program receives an error.

Normally, the system verifies that you have the required data authority when you do an input/output operation in your program. However, when you use the Open Query File (OPNQRYF) or Open Data Base File (OPNDBF) commands, the system verifies at the time the file is opened that you have the required data authority to perform the operations specified on the OPTION parameter. For more information about data authority, see "Data Authorities" on page 7-2.

The system also uses these options to determine the locks to use to protect the data integrity of the files and records being processed by your program. For more information on locks, see "Sharing Data Base Files Across Jobs" on page 8-5.

Specifying the Initial File Position

POSITION Parameter. The system needs to know where it should start processing the file after it is opened. The default is to start just before the first record in the file (the first sequential read operation will read the first record). But, you can tell the system to start at the end of the file, or at a certain record in the middle of the file by using the Override with Data Base File (OVRDBF) command. For more information about the POSITION parameter on the OVRDBF command, see the *CL Reference*. You can also dynamically set a position for the file in your program. For more information on setting position for a file in a program, see "Setting a Position in the File" on page 10-1 and your high-level language guide.

Ignoring the Keyed Sequence Access Path

ACCPTH Parameter. When you process a file with a keyed sequence access path, you normally want to use that access path to retrieve the data. The system automatically uses the keyed sequence access path if a key field is defined for the file. However, sometimes you may achieve better performance by ignoring the keyed sequence access path and processing the file in arrival sequence.

You can tell the system to ignore the keyed sequence access path in some high-level languages, or on the Open Data Base File (OPNDBF) command. When you ignore the keyed sequence access path, operations that read data by key are not allowed. Operations are done sequentially along the arrival sequence access path. (If this option is specified for a logical file with select/omit values defined, the arrival sequence access path is used and only those records meeting the select/omit values are returned to the program. The processing is done as if the DYNSLT keyword was specified for the file.)

See your high-level language guide and the OPNDBF command in the *CL Reference* for more details about ignoring the keyed sequence access path.

Note: You cannot ignore the keyed sequence access path for logical file members that are based on more than one physical file member.

Delaying End of File Processing

EOFDLY Parameter. When you are reading a data base file and your program reaches the end of the data, the system normally signals your program that there is no more data to read. Occasionally, instead of telling the program there is no more data, you might want the system to hold your program until more data arrives in the file. When more data arrives in the file, the program can read the newly arrived records. If you need that type of processing, you can use the EOFDLY parameter on the Override with Data Base File (OVRDBF) command. For more information on this parameter, see "Waiting for More Records When End of File Is Reached" on page 10-4.

Specifying the Record Length

The system needs to know the length of the record your program will be processing, but you do not have to specify record length in your program. The system automatically determines this information from the attributes and description of the file named in your program. However, as an option, you can specify the length of the record in your high-level language program. For more information on specifying record lengths in a program, see your high-level language guide.

If the file that is opened contains records that are longer than the length specified in the program, the system allocates a storage area to match the file member's record length and this option is ignored. In this case, the entire record is passed to the program. (However, some high-level languages will allow you to access only that portion of the record defined by the record length specified in the program.) If the file that is opened contains records that are less than the length specified in the program, the system allocates a storage area for the program-specified record length. The program can use the extra storage space, but only the record lengths defined for the file member are used for input/output operations.

Ignoring Record Formats

When you use a multiple format logical file, the system assumes you want to use all formats defined for that file. However, if you do not want to use all of the formats, you can specify which formats you want to use and which ones you want to ignore. If you do not use this option to ignore formats, your program can process all formats defined in the file. For more information about this processing option, see your high-level language guide.

Data Recovery and Integrity

The following section describes data integrity run-time considerations.

Protecting Your File with the Journal and Commitment Control

COMMIT Parameter. Journaling and commitment control are the preferred methods for data and transaction recovery on the AS/400 system. Data base file journaling is started by running the Start Journaling Physical File (STRJRNPF) command for the file. Access path journaling is started by running the Start Journaling Access Path (STRJRNAP) command for the access path. You tell the system that you want your files to run under commitment control through the Start Commitment Control (STRCMTCTL) command and through high-level language specifications. You can also specify the commitment control (COMMIT) parameter on the Open Data Base File (OPNDBF) and Open Query File (OPNQRYF) commands. For more information on journaling and commitment control, see Chapter 16.

Writing Data and Access Paths to Auxiliary Storage

FRCRATIO and FRCACCPTH Parameters. Normally, the AS/400 integrated data base management system determines when to write changed data from main storage to auxiliary storage. If you want to control when data base changes are written to auxiliary storage, you can use the force write ratio (FRCRATIO) parameter on either the create, change, or override data base file commands, and the force access path (FRCACCPTH) parameter on the create and change data base file commands. Using the FRCRATIO and FRCACCPTH parameters have performance and recovery considerations for your system. To understand these considerations, see Chapter 16.

Checking Changes to the Record Format Description

LVLCHK Parameter. The system checks, when you open the file, if the description of the record format you are using was changed since the program was compiled to an extent that your program cannot process the file. The system normally notifies your program of this condition. This condition is known as a level check. When you use the create or change file commands, you can specify that you want level checking. You can also override the level check attribute defined for the file by using the LVLCHK parameter on the the Override Data Base File (OVRDBF) command. For more information about this parameter, see "Effect of Changing Fields in a File Description" on page 14-1.

Checking for the File's Expiration Date

EXPDATE and EXPCHK Parameters. The system can verify that the data in the file you specify is still current. You can specify the expiration date for a file or member using the EXPDATE parameter on the create and change file commands, and you can specify whether or not the system is to check that date using the EXPCHK parameter on the Override with Data Base File (OVRDBF) command. If you do check the expiration date and the current date is greater than the expiration date, a message is sent to the system operator when the file is opened.

Preventing the Job from Changing Data in the File

INHWRT Parameter. If you want to test your program, but do not want to actually change data in files used for the test, you can tell the system to not write (inhibit) any changes to the file that the program attempts to make. To inhibit any changes to the file, specify INHWRT(*YES) on the OVRDBF command.

Sharing Data Base Files Across Jobs

By definition, all data base files can be used by many users at the same time. However, there are some operations that can lock the file, member, or data in a member that prevent it from being shared across jobs.

Each data base command or high-level language program allocates the file, member, and data that it uses. Depending on the operation requested, other users may not be able to use the allocated file, member, or records. An operation on a logical file or logical file member may allocate the file(s) and member(s) that the logical file depends on for data or an access path.

For example, the open of a logical file allocates the data of the physical file member that the logical file is based on. If the program updates the logical file member, then another user may not request, at the same time, that the physical file member used by that logical file member be cleared of data.

For a list of commonly used data base functions and the types of locks they place on data base files, see Appendix C.

Record Locks

WAITRCD Parameter. The AS/400 data base has built-in integrity for records. For example, if PGMA reads a record for update, it locks that record. Another program will not be able to read the same record for update until PGMA releases the record, but another program could read the record just for inquiry. In this way, the system ensures the integrity of the data base.

The system determines the lock condition based on the type of file processing specified in your program and the operation requested. For example, if your open options include update or delete, each record read is locked so that any number of users can read the record at the same time, but only one user can update the record.

The system normally waits a specific number of seconds for a locked record to be released before it sends your program a message that it cannot get the record you are requesting. The default record wait time is 60 seconds; however, you can set your own wait time through the WAITRCD parameter on the create and change file commands and the override data base file command. If your program is notified that the record it wants is locked by another operation, you can have your program take the appropriate action (for example, you could send a message to the operator that the requested record is currently unavailable).

The system automatically releases a lock when the locked record is updated or deleted. However, you may need to release record locks without updating the record. For information on how to release a record lock, see your high-level language guide.

Note: Using commitment control changes the record locking rules. See the *Backup and Recovery Guide* for more information on commitment control and its effect on the record locking rules.

You can use the Display Record Lock (DSPRCDLCK) command to display the current lock status (wait or held) of records for a physical file member. Depending on the parameters you specify, this command displays the lock status for a specific record or displays the lock status of all records in the member. You can also display record locks from the Work with Job (WRKJOB) display.

You can determine if your job currently has any records locked by using the Check Record Lock (CHKRCDLCK) command. This command returns a message (which you can monitor for) if your job has any locked records. The command is useful if you are using group jobs. For example, you could check to see if you had any records locked, before transferring to another group job. If you determined you did have records locked, your program could release those locks. For more information about CHKRCDLCK command, see the *CL Reference*.

File Locks

WAITFILE Parameter. Some file operations exclusively allocate the file for the length of the operation. During the time the file is allocated exclusively, any program trying to open that file will have to wait until the file is released. You may control the amount of time a program waits for the file to become available by specifying a wait time on the WAITFILE parameter of the create and change file commands and the override data base file command. If you do not specifically request a wait time, the system defaults the file wait time to zero seconds. For more information on the WAITFILE parameter, see the OVRDBF command in the *CL Reference*.

A file is exclusively allocated when an operation that changes its attributes is run. These operations (such as move, rename, grant or revoke authority, change owner, or delete) cannot be run at the same time with any other operation on the same file or on members of that file. Other file operations (such as display, open, dump, or check object) only use the file definition, and thus lock the file less exclusively. They may run at the same time with each other and with input/output operations on a member.

Member Locks

Member operations (such as add and remove) automatically allocate the file exclusively enough to prevent other file operations from occurring at the same time. Input/output operations on the same member cannot be run, but input/output operations on other members of the same file can run at the same time. In addition, clearing or reorganizing a physical file member requires exclusive allocation of the member's data.

Record Format Data Locks

RCDFMTLCK Parameter. If you want to lock the entire set of records associated with a record format (for example, all the records in a physical file), you can use the RCDFMTLCK parameter on the OVRDBF command. For more information about this parameter, see the OVRDBF command in the *CL Reference*.

Sharing Data Base Files in the Same Job

SHARE Parameter. By default, the data base management system lets one file be read and changed by many users at the same time. You may also share a file in the same job by opening the data base file more than once in the same program, or in different programs in the same job.

The SHARE parameter on the create file, change file, and override commands allows closer sharing in a job, including sharing the file, its status, its positions, and its storage area. Sharing files in the job can improve performance by reducing the amount of main storage the job needs and by reducing the time it takes to open and close the file.

Using the SHARE(*YES) parameter lets an open data path (ODP) be shared between two or more programs running in the same job. An open data path is the path through which all input/output operations for the file are performed. In a sense, it connects the program to a file. If not specified otherwise, every time a file is opened a new open data path is created. You can specify (on the create file, change file, or override commands) that, if a file is opened more than once and an open data path is still active for it in the same job, the active ODP for the file can be used with the current open of the file, and a new open data path does not have to be created. This reduces the amount of time required to open the file after the first open, and the amount of main storage required by the job. SHARE(*YES) must be specified for the first open and other opens of the same file for the open data path to be shared. A well-designed (for performance) application will normally do a shared open on files that will be opened in multiple programs in the same job.

Specifying SHARE(*NO) tells the system not to share the open data path for a file. Normally, this is specified only for those files that are seldom used or require unique processing in specific programs.

Note: A high-level language program processes an open or a close operation as though the file were not being shared. You do not specify that the file is being shared in the high-level language program. You indicate that the file is being shared in the same job through the SHARE parameter. The SHARE parameter is specified only on the create, change, and override data base file commands.

Open Considerations for Files Shared in a Job

The following items should be considered when opening a data base file that is shared in the same job.

- You must make sure that when the shared file is opened for the first time in a job, all the open options that are needed for subsequent opens of the file are specified. If the open options specified for subsequent opens of a shared file do not match those specified for the first open of a shared file, an error message is sent to the program. (You can correct this by making changes to your program or to the OPNDBF or OPNQRYF command parameters, to remove any incompatible options.)

  For example, PGMA is the first program to open FILE1 in the job and PGMA only needs to read the file. However, PGMA calls PGMB which will delete records from the same shared file. Because PGMB will delete records from the shared file, PGMA will have to open the file as if it, PGMA, is also going to delete records. You can accomplish this by using the correct specifications in the high-level language. (In order to accomplish this in some high-level languages, you may have to use file operation statements that are never run. See your high-level language guide for more details.) You can also specify the file processing option on the OPTION parameter on the Open Data Base File (OPNDBF) and Open Query File (OPNQRYF) commands.
- Sometimes sharing a file within a job is not possible. For example, one program may need records from a file in arrival sequence and another program needs the records in keyed sequence. In this situation, you cannot share the open data path. You would specify SHARE(*NO) on the Override with Data Base File (OVRDBF) command to ensure that the file was not shared within the job.
- If debug mode is entered with UPDPROD(*NO) after the first open of a shared file in a production library, subsequent shared opens of the file share the original open data path and allow the file to be changed. To prevent this, specify SHARE(*NO) on the OVRDBF command before opening files being debugged.
- The use of commitment control for the first open of a shared file, requires that all subsequent shared opens also use commitment control. (See the *Backup and Recovery Guide* for more information on commitment control.)
- If you did not specify a library name in the program or on the Override with Data Base File (OVRDBF) command (*LIBL is used), the system assumes that the library list has not changed since the last open of the same shared file with *LIBL specified. If the library list has changed, you should specify the library name on the OVRDBF command to ensure that the correct file is opened.
- On subsequent opens of a shared file, if the record length is specified in the program and it is larger than the record length specified in the first open of the file, an error message (MCH0601) is sent to the program.
- Overrides and program specifications specified on the first open of the shared file are processed. Overrides and program specifications specified on subsequent opens, other than those that change the file name or the value specified on the SHARE or LVLCHK parameters on the OVRDBF command, are ignored.
- Overrides specified for a first open using the OPNQRYF command can be used to change the names of the files, libraries, and members that should be processed by the Open Query File command. Any parameter values specified on the Override with Data Base command other than TOFILE, MBR, LVLCHK, and SEQONLY are ignored by the OPNQRYF command.

Note: The Open Query File (OPNQRYF) command never shares an existing shared open data path in the job. If there is already a shared ODP in the job with the same file, library, and member name as the one specified on the Open Query

File command, the system sends an error message and the query file is not opened.

Input/Output Considerations for Files Shared in a Job

The following items should be considered when processing a data base file that is shared in the same job.

- Because there is only one open data path for a shared file, there is only one record position maintained for all programs in the job that are sharing the file. If a program establishes a position for a record by using a read or a read-for-update operation, then calls another program that also uses the shared file, the record position may have moved or a record lock been released when the called program returns to the calling program. This can cause errors in the calling program because of an unexpected record position or lock condition. When sharing files, it is your responsibility to manage the record position and record locking considerations by re-establishing position and locks.
- If a shared file is first opened for update, this does not necessarily cause every subsequent program that shares the file to request a record lock. The system determines the type of record lock needed for each program using the file. The system tries to keep lock contention to a minimum, while still ensuring data integrity.

  For example, PGMA is the first program in the job to open a shared file. PGMA intends to update records in the file; therefore, when the program reads a record for update, it will lock the record. PGMA then calls PGMB. PGMB also uses the shared file, but it does not update any records in the file; PGMB just reads records. Even though PGMA originally opened the shared file as update-capable, PGMB will not lock the records it reads, because of the processing specifications in PGMB. Thus, the system ensures data integrity, while minimizing record lock contention.

Close Considerations for Files Shared in a Job

The following items should be considered when closing a data base file that is shared in the same job.

- The complete processing of a close operation, which includes releasing file, member, and record locks; forcing changes to auxiliary storage; and destroying the open data path, is done only when the last program to open the shared open data path closes it.
- If the file was opened with the Open Data Base File (OPNDBF) or the Open Query File (OPNQRYF) command, use the Close File (CLOF) command to close the file. A Reclaim Resources (RCLRSC) command may also be used to close the file, if TYPE(*NORMAL) is specified or allowed to default on the OPNDBF or OPNQRYF command.

Examples of Closing Shared Files

The following examples show some of the things to consider when closing a file that is shared in the same job.

Example 1: Using a single set of files with similar processing options.

In this example, the user signs on and most of the programs that are used process the same set of files.

A CL program (PGMA) is used as the first program (to set up the application, including overrides and opening the shared files). PGMA then transfers control to

PGMB, which displays the application menu. Assume, in this example, that files A, B, and C are used, and files A and B are to be shared. Files A and B were created with SHARE(*NO); therefore an OVRDBF command should precede each of the OPNDBF commands to specify the SHARE(*YES) option. File C was created with SHARE(*NO) and File C is not to be shared in this example.

```
PGMA:   PGM       /* PGMA - Initial program  */
        OVRDBF    FILE(A) SHARE(*YES)
        OVRDBF    FILE(B) SHARE(*YES)
        OPNDBF    FILE(A) OPTION(*ALL) ....
        OPNDBF    FILE(B) OPTION(*INP) ...
        TFRCTL    PGMB
        ENDPGM


PGMB:   PGM       /* PGMB - Menu program */
        DCLF      FILE(DISPLAY)
BEGIN:  SNDRCVF   RCDFMT(MENU)
        IF        (&RESPONSE *EQ '1') CALL PGM11
        IF        (&RESPONSE *EQ '2') CALL PGM12
        .
        .
        IF        (&RESPONSE *EQ '90') SIGNOFF
        GOTO      BEGIN
        ENDPGM
```

In this example, assume that:

- PGM11 opens files A and B. Because these files were opened as shared by the OPNDBF commands in PGMA, the open time is reduced. The close time is also reduced when the shared open data path is closed. The Override with Data Base File (OVRDBF) commands remain in effect even though control is transferred (with the Transfer Control [TFRCTL] command in PGMA) to PGMB.
- PGM12 opens files A, B, and C. File A and B are already opened as shared and the open time is reduced. Because file C is used only in this program, the file is not opened as shared.

In this example, the Close File (CLOF) was not used because only one set of files is required. When the operator signs off, the files are automatically closed. It is assumed that PGMA (the initial program) is called only at the start of the job.

Note: The display file (DISPLAY) in PGMB can also be specified as a shared file, which would improve the performance for opening the display file in any programs that use it later.

In Example 1, the OPNDBF commands are placed in a separate program (PGMA) so that the other processing programs in the job run as efficiently as possible. That is, the important files used by the other programs in the job are opened in PGMA. After the files are opened by PGMA, the main processing programs (PGMB, PGM11, and PGM12) can share the files; therefore, their open and close requests will be faster. In addition, by placing the open commands (OPNDBF) in PGMA rather than in PGMB, the amount of main storage used for PGMB is reduced.

Any overrides and opens can be specified in the initial program (PGMA); then that program can be removed from the job (for example, by transferring out of it). However, the open data paths that the program created when it opened the files remain in existence and can be used by other programs in the job.

Note the handling of the OVRDBF commands in relation to the OPNDBF commands. Overrides must be specified before the file is opened. Some of the parameters on the OVRDBF command also exist on the OPNDBF command. If there are conflicts, the OVRDBF value is used.

Example 2: Using multiple sets of files with similar processing options.

Assume that a menu requests the operator to specify the application program (for example, accounts receivable or accounts payable) that uses the Open Data Base File (OPNDBF) command to open the required files. When the application is ended, the Close File (CLOF) command closes the files. The CLOF command is used to help reduce the amount of main storage needed by the job. In this example, different files are used for each application. The user normally works with one application for a considerable length of time before selecting a new application.

An example of the accounts receivable programs follows:

```
PGMC:   PGM       /* PGMC PROGRAM */
        DCLF      FILE(DISPLAY)
BEGIN:  SNDRCVF   RCDFMT(TOPMENU)
        IF        (&RESPONSE *EQ '1') CALL ACCRECV
        IF        (&RESPONSE *EQ '2') CALL ACCPAY
        .
        .
        IF        (&RESPONSE *EQ '90') SIGNOFF
        GOTO      BEGIN
        ENDPGM



ACCREC: PGM       /* ACCREC PROGRAM */
        DCLF      FILE(DISPLAY)
        OVRDBF    FILE(A) SHARE(*YES)
        OVRDBF    FILE(B) SHARE(*YES)
        OPNDBF    FILE(A) OPTION(*ALL) ....
        OPNDBF    FILE(B) OPTIONS(*INP) ...
BEGIN:  SNDRCVF   RCDFMT(ACCRMENU)
        IF        (&RESPONSE *EQ '1') CALL PGM21
        IF        (&RESPONSE *EQ '2') CALL PGM22
        .
        .
        IF        (&RESPONSE *EQ '88') DO /* Return */
                  CLOF FILE(A)
                  CLOF FILE(B)
                  RETURN
                  ENDDO
        GOTO      BEGIN
        ENDPGM
```

The program for the accounts payable menu would be similar, but with a different set of OPNDBF and CLOF commands.

For this example, files A and B were created with SHARE(*NO). Therefore, an OVRDBF command must precede the OPNDBF command. As in Example 1, the amount of main storage used by each job could be reduced by placing the OPNDBF commands in a separate program and calling it. A separate program could also be created for the CLOF commands. The OPNDBF commands could be placed in an application setup program that is called from the menu, which transfers control to

the specific application program menu (any overrides specified in this setup program are kept). However, calling separate programs for these functions also uses system resources and, depending on the frequency with which the different menus are used, it may be better to include the OPNDBF and CLOF commands in each application program menu as shown in this example.

Another choice is to use the Reclaim Resources (RCLRSC) command in PGMC (the setup program) instead of using the Close File (CLOF) commands. The RCLRSC command closes any files and frees any leftover storage associated with any files and programs that were called and have since returned to the calling program. However, RCLRSC will *not* close files opened with TYPE(*PERM) specified on the Open Data Base File (OPNDBF) or Open Query File (OPNQRYF) commands.

The following example shows the RCLRSC command used to close files:

```
.
.
IF          (&RESPONSE *EQ '1') DO
            CALL ACCRECV
            RCLRSC
            ENDDO
IF          (&RESPONSE *EQ '2') DO
            CALL ACCPAY
            RCLRSC
            ENDDO
.
.
```

Example 3: Using a single set of files with different processing requirements.

If some programs need read-only file processing and others need some or all of the options (input/update/add/delete), then one of the following methods can be used. The same methods apply if a file is to be processed with certain command parameters in some programs and not in others (for example, sometimes the commit option should be used).

A single Open Data Base File (OPNDBF) command could be used to specify OPTION(*ALL) and the open data path would be opened shared (if, for example, a previous OVRDBF command was used to specify SHARE(*YES)). Each program could then open a subset of the options. The program requests the type of open depending on the specifications in the program. In some cases this does not require any more considerations because a program specifying an open for input only would operate similarly as if it had not done a shared open (for example, no additional record locking occurs when a record is read).

However, some options specified on the OPNDBF command may affect how the program operates. For example, SEQONLY(*NO) is specified on the open command for a file in the program. An error would occur if the OPNDBF command used SEQONLY(*YES) and a program attempted an operation that was not valid with sequential-only processing.

The ACCPTH parameter must also be consistent with the way programs will use the access path (arrival or keyed).

If COMMIT(*YES) is specified on the Open Data Base File (OPNDBF) command and the Start Commitment Control (STRCMTCTL) command specifies LCKLVL(*ALL), any read operation of a record will lock that record (per commitment control record locking

rules). This may cause records to be locked unexpectedly and cause errors in the program. For more information about commitment control and how it effects the locking of records, see the *Backup and Recovery Guide*.

Two OPNDBF commands could be used for the same data (for example, one with OPTION(*ALL) and the other specifying OPTION(*INP)). The second use must be a logical file pointing to the same physical file(s). This logical file can then be opened as SHARE(*YES) and multiple uses made of it during the same job.

Sequential-Only Processing

SEQONLY and NBRRCDS Parameters. If your program processes a data base file sequentially for input or output, you might be able to improve performance by using the sequential-only processing (SEQONLY) parameter on the Override with Data Base File (OVRDBF) or the Open Data Base File (OPNDBF) commands. (The Open Query File [OPNQRYF] command uses sequential-only processing whenever possible.) Depending on your high-level language specifications, the high-level language may also default to sequential-only processing. For example, if you open a file for input only and the only file operations specified in the high-level language program are sequential read operations, then the high-level language automatically requests sequential-only processing.

Note: File positioning operations are not considered sequential read operations; therefore, a high-level language program containing positioning operations will *not* automatically request sequential-only processing. (The SETLL operation in RPG/400 and the START operation in COBOL/400 are examples of file positioning operations.) Even though the high-level language program may not automatically request sequential-only processing, you may request it using the SEQONLY parameter on the OVRDBF command.

If you specify sequential-only processing, you can also specify the number of records to be moved as one unit between the system data base main storage area and the job's internal data main storage area. If you do not specify the sequential-only number of records to be moved, the system calculates a number based on the number of records that fit into a 4096-byte buffer.

The system also provides you a way to control the number of records that are moved as a unit between auxiliary storage and main storage. If you are reading the data in the file in the same order as the data is physically stored, you may be able to improve the performance of your job by using the NBRRCDS parameter on the OVRDBF command.

For more information about the SEQONLY and the NBRRCDS parameters, see the OVRDBF command in the *CL Reference*.

Open Considerations for Sequential-Only Processing

The following considerations apply for opening files when sequential-only processing is specified. If the system determines that sequential-only processing is not allowed, a message is sent to the program to indicate that the request for sequential-only processing is not being accepted; however, the file is still opened for processing.

- If the program opened the member for output only, and if SEQONLY(*YES) was specified (number of records was not specified), and either the opened member is a logical member, a uniquely keyed physical member, or there are other access paths to the physical member, SEQONLY(*YES) is changed to SEQONLY(*NO) so the

program can handle possible errors (for example, duplicate keys, conversion mapping, and select/omit errors) at the time of the output operation. If you want the system to run sequential-only processing, change the SEQONLY parameter to include both the *YES value and number of records specification.

- Sequential-only processing can be specified only for input-only or output-only (add) operations. If the program specifies update or delete operations, sequential-only processing is not allowed by the system.
- If a file is being opened for output, it must be a physical file or a logical file based on one physical file member.
- If the record is opened for input-only, the record length plus the feedback area must be less than or equal to 32 767 bytes. (The feedback area size is 10 + key length * number of records specified on the SEQONLY parameter.)
- The member cannot be opened for commitment control; otherwise, sequential-only processing is not allowed by the system.
- For output-only, the number of records specified to be moved as a unit and the force ratio are compared and automatically adjusted as necessary. If the number of records is larger than the force ratio, the number of records is reduced to equal the force ratio. If the opposite is true, the force ratio is reduced to equal the number of records.

The following considerations apply when sequential-only processing is not specified and the file is opened using the Open Query File (OPNQRYF) command. If these conditions are satisfied, a message is sent to indicate that sequential-only processing will be performed and the query file is opened.

- If the OPNQRYF command specifies the name of one or more fields on the group field (GRPFLD) parameter, or OPNQRYF requires group processing.
- If the OPNQRYF command specifies one or more fields, or *ALL on the UNIQUEKEY parameter.
- If a view is used with the DISTINCT option on the SQL/400 SELECT statement, then SEQONLY(*YES) processing is automatically performed.

For more details about the OPNQRYF command, see "Using the Open Query File (OPNQRYF) Command" on page 9-2.

Input/Output Considerations for Sequential-Only Processing

The following considerations apply for input/output operations on files when sequential-only processing is specified.

- For input, your program receives one record at a time from the input buffer. When all records in the input buffer are processed, the system automatically reads the next set of records.
- For output, your program must move one record at a time to the output buffer. When the output buffer is full, the system automatically adds the records to the data base.

  Note: If you are using a journal, the entire buffer is written to the journal at one time as if the entries had logically occurred together.

  If you use sequential-only processing for output, you might not see all the changes made to the file as they occur. For example, if sequential-only is specified for a file being used by PGMA and PGMA is adding new records to the file and the SEQONLY parameter was specified with 5 as the number of records in the buffer, then only when the buffer is filled will the newly added records be transferred to the data base. In this example, only when the fifth record was added, would the first five records be transferred to the data base, and be available for processing by other jobs in the system.

In addition, if you use sequential-only processing for output, some additions might not be made to the data base if you do not handle the errors that could occur when records are moved from the buffer to the data base. For example, assume the buffer holds five records, and the third record in the buffer had a key that was a duplicate of another record in the file and the file was defined as a unique-key file. In this case, when the system transfers the buffer to the data base it would add the first two records and then get a duplicate key error for the third. Because of this error, the third, fourth, and fifth records in the buffer would *not* be added to the data base.

- The force-end-of-data function can be used for output operations to force all records in the buffer to the data base (except those records that would cause a duplicate key in a file defined as having unique keys, as described previously). The force-end-of-data function is only available in certain high-level languages. See your high-level language guide for more details.

Close Considerations for Sequential-Only Processing

When a file for which sequential-only processing is specified is closed, all records still in the output buffer are added to the data base. However, if an error occurs for a record, any records following that record are not added to the data base.

If multiple programs in the same job are sharing a sequential-only output file, the output buffer is not emptied until the final close occurs. Consequently, a close (other than the last close in the job) does not cause the records still in the buffer to appear in the data base for this or any other job.

Run Time Summary

The following tables list parameters that control your program's use of the data base file member, and indicates where these parameters can be specified. For parameters that can be specified in more than one place, the system merges the values. The Override with Data Base File (OVRDBF) command parameters take precedence over program parameters, and Open Data Base File (OPNDBF) or Open Query File (OPNQRYF) command parameters take precedence over create or change file parameters.

Note: Any override parameters other than TOFILE, MBR, LVLCHK, SEQONLY, and INHWRT are ignored by the OPNQRYF command.

Refer to your high-level language guide for more information about the program processing parameters, and the *CL Reference* for more information about the command processing parameters.

Description	Parameter	CRTPF or CRTLF Cmd	CHGPF or CHGLF Cmd	OPNDBF Cmd	OPNQRYF Cmd	OVRDBF Cmd
File name	FILE	X	X[1]	X	X	X
Library name		X	X[2]	X	X	X
Member name	MBR	X		X	X	X
Member processing options	OPTION			X	X	
Record format lock state	RCDFMTLCK					X
Starting file position after open	POSITION					X
Program performs only sequential processing	SEQONLY			X	X	X
Ignore keyed access path	ACCPTH			X		
Time to wait for file locks	WAITFILE	X	X			X
Time to wait for record locks	WAITRCD	X	X			X
Prevent overrides	SECURE					X
Number of records to be transferred from auxiliary to main storage	NBRRCDS					X
Share open data path with other programs	SHARE	X	X			X
Format selector	FMTSLR	X[3]	X[3]			X
Force ratio	FRCRATIO	X	X			X
Inhibit write	INHWRT					X
Level check record formats	LVLCHK	X	X			X
Expiration date checking	EXPCHK					X
Expiration date	EXPDATE	X[4]	X[4]			X
Force access path	FRCACCPTH	X	X			
Commitment control	COMMIT			X	X	
End-of-file delay	EOFDLY					X

[1] File name: The CHGPF and CHGLF commands use the file name for identification only. You cannot change the file name.

[2] Library name: The CHGPF and CHGLF commands use the library name for identification only. You cannot change the library name.

[3] Format selector: Used on the CRTLF and CHGLF commands only.

[4] Expiration date: Used on the CRTPF and CHGPF commands only.

Figure 8-1. Data Base Processing Options Specified on CL Commands

Description	RPG/ 400	COBOL/ 400	AS/400 BASIC	AS/400 PL/I	AS/400 Pascal
File name	X	X	X	X	X
Library name			X	X	X
Member name			X	X	X
Program record length	X	X	X	X	X
Member processing options	X	X	X	X	X
Record format lock state			X	X	
Record formats the program will use	X		X		
Clear physical file member of records		X	X		X
Program performs only sequential processing	X	X		X	X
Ignore keyed access path	X	X	X	X	X
Share open data path with other programs				X	X
Level check record formats	X	X	X	X	
Commitment control	X	X		X	

Note: Control language (CL) programs can also specify many of these parameters. See Figure 8-1 on page 8-16 for more information about the data base processing options that can be specified on CL commands.

Figure 8-2. Data Base Processing Options Specified In Programs

Chapter 9. Opening a Data Base File

This chapter discusses opening a data base file. In addition, the CL commands Open Data Base File (OPNDBF) and Open Query File (OPNQRYF) are discussed.

Opening a Data Base File Member

To use a data base file in a program, your program must issue an open operation to the data base file. (If you do not specify an open operation in some programming languages, they automatically open the file for you. See your high-level language guide for more information about opening data base files.) If you did not specify a member name in your program or on an Override with Data Base File (OVRDBF) command, the first member (as defined by creation date and time) in the file is used. After finding the member, the system connects your program to the data base file. This allows your program to perform input/output operations to the file. For more information about opening files in your high-level language program, see the appropriate high-level language guide.

You can open a data base file with statements in your high-level language program. You can also use the CL open commands: Open Data Base File (OPNDBF) and Open Query File (QPNQRYF). The OPNDBF command is useful in an initial program in a job for opening shared files. The OPNQRYF command is very effective in selecting and arranging records outside of your program. Then, your program can use the information supplied by the OPNQRYF command to process only the data it needs.

Using the Open Data Base File (OPNDBF) Command

Usually, when you use the OPNDBF command, you can use the defaults for the command parameter values. In some instances you may want to specify particular values, instead of using the default values, for the following parameters:

OPTION Parameter. Specify the *INP option if your application programs uses input-only processing (reading records without updating records). This will allow the system to read records without trying to lock each one for possible update. Specify the *OUT option if your application programs uses output-only processing (writing records into a file but not reading or updating existing records).

Note: If your program does direct output operations to active records (updating by relative record number), *ALL must be specified instead of *OUT. If your program does direct output operations to deleted records only, *OUT must be specified.

MBR Parameter. If a member, other than the first member in the file, is to be opened, you must specify the name of the member to be opened or issue an Override with Data Base File (OVRDBF) command before the Open Data Base File (OPNDBF) command.

Note: You must specify a member name on the OVRDBF command to use a member (other than the first member) to open in subsequent programs.

OPNID Parameter. If an identifier other than the file name is to be used, you must specify it. The open identifier may be used in other CL commands to process the

file. For example, the Close File (CLOF) command uses the identifier to specify which file is to be closed.

ACCPTH Parameter. If the file has a keyed sequence access path and either (1) the open option is *OUT, or (2) the open option is *INP or *ALL, but your program does not use the keyed access path, then you can specify ACCPTH(*ARRIVAL) on the OPNDBF parameter. Ignoring the keyed access path may improve your job's performance.

SEQONLY Parameter. Specify *YES if subsequent application programs process the file sequentially. This parameter can also be used to specify the number of records that should be transferred between the system data buffers and the program data buffers. SEQONLY(*YES) is not allowed unless OPTION(*INP) or OPTION(*OUT) is also specified on the Open Data Base File (OPNDBF) command.

COMMIT Parameter. Specify *YES if the application programs use commitment control. If you specify *YES you must be running in a commitment control environment (the Start Commitment Control [STRCMTCTL] command was processed) or the OPNDBF command will fail. Use the default of *NO if the application programs do not use commitment control.

TYPE Parameter. Specify what you wish to happen when unmonitored exceptions occur in your application program. If you specify *NORMAL, your program can issue a Reclaim Resources (RCLRSC) command to close the files, or the high-level language you are using may perform a close operation. Specify *PERM if you want to continue the application without opening the files again. Any error message received in your program when files are open cause those files to be closed if TYPE(*NORMAL) was specified. TYPE(*PERM) allows the files to remain open even if an error message is received.

For more information about this command, see OPNDBF in the *CL Reference*.

Using the Open Query File (OPNQRYF) Command

The Open Query File (OPNQRYF) command is a CL command that allows you to perform many data processing functions on data base files. Essentially, the OPNQRYF command acts as a filter between the processing program and the data base records. The data base file can be a physical or logical file.

Unlike a data base file created with the Create Physical File (CRTPF) command or the Create Logical File (CRTLF) command, the OPNQRYF command creates only a temporary file for processing the data, it does not create a permanent file.

To understand the OPNQRYF command support, you should already be familiar with data base concepts such as physical and logical files, key fields, record formats, and join logical files. These concepts are discussed in Part 1 and Part 2.

The OPNQRYF command has functions similar to those in DDS, and the CRTPF and CRTLF commands. DDS requires source statements and a separate step to create the file. OPNQRYF allows a dynamic definition without using DDS. The OPNQRYF command does not support all of the DDS functions, but it supports significant functions that go beyond the capabilities of DDS.

In addition, Query can be used to perform some of the function the OPNQRYF command performs. However, the OPNQRYF command is more useful as a programmer's tool.

The OPNQRYF command parameters also have many functions similar to the SQL/400 SELECT functions. For example, the FILE parameter is similar to the SQL/400 FROM statement, the QRYSLT parameter is similar to the SQL/400 WHERE statement, the GRPFLD parameter is similar to the SQL/400 GROUP BY statement, and the GRPSLT parameter is similar to the SQL/400 HAVING statement. For more information about SQL/400, see the *SQL/400 Programmer's Guide*.

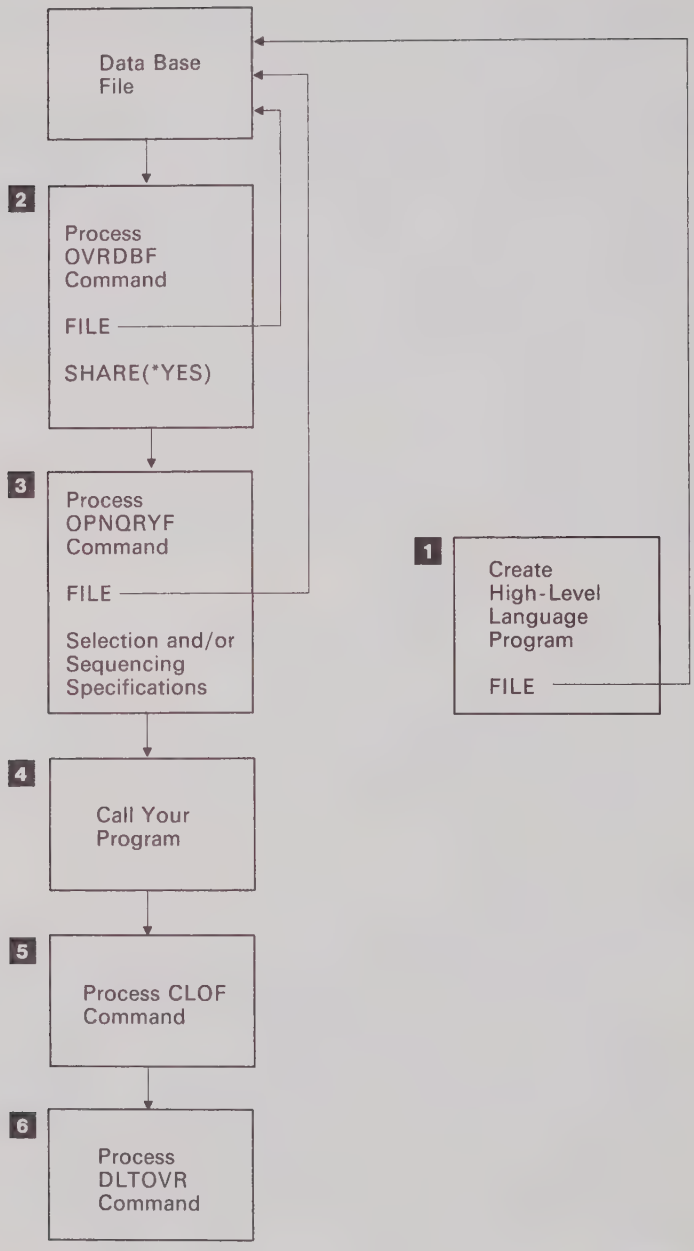
The following is a list of the major functions supplied by OPNQRYF. Each of these functions is described later in this section.

- Dynamic record selection
- Dynamic keyed access path
- Dynamic keyed access path over a join
- Dynamic join
- Handling missing records in secondary join files
- Unique-key processing
- Mapped field definitions
- Group processing
- Final total-only processing
- Optimizing performance

To understand the OPNQRYF command, you must be familiar with its two processing approaches: using a format in the file, and using a file with a different format. The typical use of the OPNQRYF command is to select, arrange, and format the data so it can be read sequentially by your high-level language program.

Using an Existing Record Format in the File

Assume you only want your program to process the records in which the *Code* field is equal to D. You create the program as if there were only records with a D in the *Code* field. That is, you do not code any selection operations in the program. You then run the OPNQRYF command, and specify that only the records with a D in the *Code* field are to be returned to the program. The OPNQRYF command does the record selection and your program processes only the records that meet the selection criteria. You can use this approach to select a set of records, return records in a different sequence than they are stored, or both. The following is an example of using the OPNQRYF command to select and sequence records:



1 Create the high-level language program to process the data base file as you would any normal program using externally described data. Only one format can be used, and it must exist in the file.

2 Run the OVRDBF command specifying the file and member to be processed and SHARE(*YES). (If the member is permanently changed to SHARE(*YES) and the first or only member is the one you want to use, this step is not necessary.)

The OVRDBF command can be run after the OPNQRYF command, unless you want to override the file name specified in the OPNQRYF command. In this discussion and in the examples, the OVRDBF command is shown first.

There are restrictions on using the OVRDBF command with the OPNQRYF command. For example, MBR(*ALL) causes an error message and the file is not opened. Refer to "Considerations for Files Shared in a Job" on page 9-36 for more information.

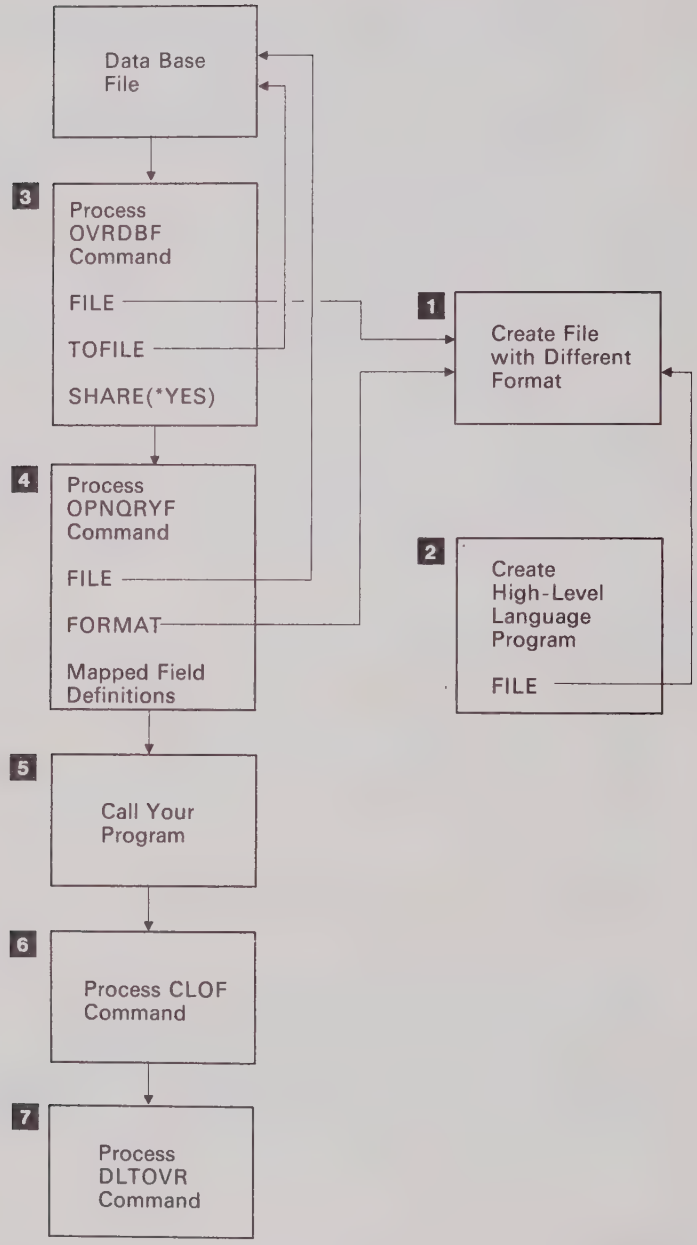
3. Run the OPNQRYF command, specifying the data base file, member, and format names and any selection and/or sequencing options.
4. Call the high-level language program you created in step 1. Only a high-level language program can use the file created by the OPNQRYF command. CL commands (for example, the Copy File [CPYF] and the Display Physical File Member [DSPPFM] commands) and utilities (for example, Query) do not work with the OPNQRYF command.
5. Close the file that you opened in step 3, unless you want the file to remain open. The Close File (CLOF) command can be used to close the file.
6. Delete the override that was specified in step 2 with the the Delete Override (DLTOVR) command. It may not always be necessary to delete the override, but the command is shown in all the examples for consistency.

Using a File with a Different Record Format

For more advanced functions of the Open Query File (OPNQRYF) command (such as dynamically joining records from different files), you must define a new file that contains a different record format. This new file is a separate file from the one you are going to process. This new file contains the fields that you want to create with the OPNQRYF command. This powerful capability also lets you define fields that do not currently exist in your data base records, but can be derived from them.

When you code your high-level language program, specify the name of the file with the different format so the externally described field definitions of both existing and derived fields can be processed by the program.

Before calling your high-level language program, you must specify an Override with Data Base File (OVRDBF) command to direct your program file name to the open query file. On the OPNQRYF command, specify both the data base file and the new file with the special format to be used by your high-level language program. If the file you are querying does not have SHARE(*YES) specified, you must specify SHARE(*YES) on the OVRDBF command.



1 Specify the DDS for the file with the different record format, and create the file. This file contains the fields that you want to process with your high-level language program. There is normally no data in this file, and it does not require a member. You normally create this file as a physical file without keys. A field reference file can be used to describe the fields. The record format name can be different from the record format name in the data base file that is specified. You can use any data base or DDM file for this function. The file could be a logical file and it could be indexed. It could have one or more members, with or without data.

2. Create the high-level language program to process the file with the record format that you created in step 1. In this program, do not name the data base file that contains the data.
3. Run the Override with Data Base File (OVRDBF) command. Specify the name of the file with the different (new) record format on the FILE parameter. Specify the name of the data base file that you want to query on the TOFILE parameter. You may also specify a member name on the MBR parameter. If the data base member you are querying does not have SHARE(*YES) specified, you must also specify SHARE(*YES) on the OVRDBF command.
4. Run the Open Query File (OPNQRYF) command. Specify the data base file to be queried on the FILE parameter, and specify the name of the file with the different (new) format that was created in step 1 on the FORMAT parameter. Mapped field definitions may be required on the OPNQRYF command to describe how to map the data from the data base file into the format that was created in step 1. You may also specify selection and sequencing options.
5. Call the high-level language program you created in step 2.
6. The first file named in step 4 for the FILE parameter was opened with OPNQRYF as SHARE(*YES) and is still open. The file must be closed. The Close File (CLOF) command can be used.
7. Delete the override that was specified in step 3.

The previous steps show the normal flow using externally described data. It is not necessary to create unique DDS and record formats for each OPNQRYF command. You may reuse an existing record format. However, all fields in the record format must be actual fields in the real data base file or defined by mapped field definitions. If you use program-described data, you can create the program at any time.

You can use the file created in step 1 to hold the data created by the Open Query File (OPNQRYF) command. For example, you can replace step 5 with a high-level language processing program that copies data to the file with the different format (the CPYF command cannot be used). You can then follow step 5 with the CPYF command or Query.

Examples

The following sections describe how to specify both the OPNQRYF parameters for each of the major functions discussed earlier and how to use the Open Query File command with your high-level language program.

Notes:

1. If you are running the OPNQRYF command from a command entry line, any error messages occurring after the OPNQRYF command is successfully run will close the file unless TYPE(*PERM) was specified on the OPNQRYF command. The system automatically runs a Reclaim Resources (RCLRSC) command if an error message occurs. You must run the OPNQRYF command again after the error is removed.

   Note: If the OPNQRYF command specified TYPE(*PERM), the file is not automatically closed by the system.

2. After running a program that uses the Open Query File command for sequential processing, the file position is normally at the end of the file. If you want to run the same program or a different program with the same files, you must position the file or close the file and open it with the same OPNQRYF command. You

may position the file with the Position Data Base File (POSDBF) command. In some cases, a high-level language program statement can be used.

Selecting Records without Using DDS

Dynamic record selection allows you to request a subset of the records in a file without using DDS. For example, you can select records that have a specific value or range of values (for example, all customer numbers between 1000 and 1050). The Open Query File (OPNQRYF) command allows you to combine these and other selection functions to produce powerful record selection capabilities.

Examples of Selecting Records Using the Open Query File (OPNQRYF) Command

In all of the following examples, it is assumed that a single-format data base file (physical or logical) is being processed. (The FILE parameter on the OPNQRYF command allows you to specify a record format name if the file is a multiple format logical file.)

Example 1: Selecting records with a specific value

Assume you want to select all the records from FILEA where the value of the *Code* field is D. Your processing program is PGMB. PGMB only sees the records that meet the selection value (you do not have to test in your program).

Note: You can specify parameters easier by using the prompt function for the OPNQRYF command. For example, you can specify an expression for the QRYSLT parameter without the surrounding delimiters because the system will add the apostrophes.

Specify the following:

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('CODE *EQ "D" ')
CALL       PGM(PGMB)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

Notes:

1. The entire expression in the QRYSLT parameter must be enclosed in apostrophes.
2. When specifying field names in the OPNQRYF command, the names in the record are not enclosed in apostrophes.
3. Character literals must be enclosed by quotation marks or two apostrophes. (The quotation mark character is used in the examples.) It is important to place the character(s) between the quotation marks in either uppercase or lowercase to match the value you want to find in the data base. (The examples are all shown in uppercase.)
4. To request a selection against a numeric constant, specify:

   ```
   OPNQRYF    FILE(FILEA) QRYSLT('AMT *GT 1000.00')
   ```

   Notice that numeric constants are *not* enclosed by two apostrophes (quotation marks).

When comparing two fields or constants, the data types must be compatible. The following table describes the valid comparisons.

	Any Numeric	Character
Any Numeric	Valid	Not Valid
Character	Not Valid	Valid

Figure 9-1. Valid Data Type Comparisons for the OPNQRYF command

Note: For DBCS information, see Appendix B.

The performance of record selection can be greatly enhanced if some file on the system uses the field being selected in a keyed access path. This allows the system to quickly access only the records that meet the selection values. If no such access path exists, the system must read every record to determine if it meets the selection values.

Even if an access path exists on the field you want to select from, the system may not use the access path. For example, if it is faster for the system to process the data in arrival sequence, it will do so. See the discussion in "Performance Considerations" on page 9-33 for more details.

Example 2: Selecting records with a specific date value

Assume you want to process all records in which the *Date* field in the record is the same as the current date. Also assume the *Date* field is in the same format as the system date. In a CL program, you can specify:

```
DCL        VAR(&CURDAT)  TYPE(*CHAR)  LEN(6)
RTVSYSVAL  SYSVAL(QDATE) RTNVAR(&CURDAT)
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('"' *CAT &CURDAT *CAT '" *EQ DATE')
CALL       PGM(PGMB)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

A CL variable is assigned with a leading ampersand (&) and is not enclosed in apostrophes. The quotation mark is enclosed in apostrophes as is the string *EQ DATE. The quotation mark is concatenated to the value of the CL variable, as is the remainder of the QRYSLT string.

It is important to know whether the data in the data base is defined as character or numeric. In this example, the *Date* field is assumed to be character. If the field were defined as numeric and you were using a constant, the QRYSLT statement would be specified as follows:

```
QRYSLT('123187 *EQ DATE')
```

Note that double apostrophes are not used around the constant. If you replace the constant with a variable, the QRYSLT parameter would be written as follows (see the difference from the one specified above):

```
QRYSLT(&CURDAT *CAT ' *EQ DATE')
```

If a numeric field exists in the data base and you want to compare it to a variable, only a character variable can be used. For example, to select all records where a packed *Date* field is greater than a variable, you must ensure the variable is in character form. Normally, this will mean that before the Open Query File (OPNQRYF) command, you use the Change Variable (CHGVAR) command to change the variable from a decimal field to a character field. The CHGVAR command would be specified as follows:

```
CHGVAR VAR(&CHARVAR)  VALUE('123188')
```

The QRYSLT parameter would be specified as follows:

```
QRYSLT(&CHARVAR  *CAT ' *GT DATE')
```

Example 3: Selecting records in a range of values

Assume you have a *Date* field specified in the character format YYMMDD, and you want to process all records for 1988. You can specify:

```
OVRDBF      FILE(FILEA) SHARE(*YES)
OPNQRYF     FILE(FILEA) QRYSLT('DATE *EQ %RANGE("880101" +
               "881231") ')
CALL        PGM(PGMC)
CLOF        OPNID(FILEA)
DLTOVR      FILE(FILEA)
```

If the ranges are variables, you would specify the QRYSLT parameter as follows:

```
QRYSLT('DATE *EQ %RANGE("' *CAT &LORNG *CAT'"' *BCAT '"'  +
   *CAT &HIRNG *CAT '")')
```

Example 4: Selecting records using the contains function

Assume you want to process all records in which the *Addr* field contains the street named BROADWAY. The contains (*CT) function determines if the characters appear anywhere in the named field. You can specify:

```
OVRDBF      FILE(FILEA) SHARE(*YES)
OPNQRYF     FILE(FILEA) QRYSLT('ADDR *CT "BROADWAY" ')
CALL        PGM(PGMC)
CLOF        OPNID(FILEA)
DLTOVR      FILE(FILEA)
```

In this example, assume that the data is in uppercase in the data base record. If the data were in lowercase or mixed case, you could specify a translation function to translate the lowercase or mixed case data to uppercase before the comparison is made. The system-provided table QSYSTRNTBL translates the letters a through z to uppercase. (You could use any translation table to perform the translation.) Therefore, you can specify:

```
OVRDBF      FILE(FILEA) SHARE(*YES)
OPNQRYF     FILE(FILEA) QRYSLT('%XLATE(ADDR QSYSTRNTBL) *CT +
               "BROADWAY" ')
CALL        PGM(PGMC)
CLOF        OPNID(FILEA)
DLTOVR      FILE(FILEA)
```

When the %XLATE function is used on the QRYSLT statement, the value of the field passed to the high-level language program appears as it is in the data base. You can force the field to appear in uppercase by using the %XLATE function on the MAPFLD parameter.

Example 5: Selecting records using multiple fields

Assume you want to process all records in which the *Amt* field is equal to zero or the *Lstdat* field (YYMMDD order in character format) is equal to or less than 881231. You can specify:

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('AMT *EQ 0 *OR LSTDAT +
                *LE "881231" ')
CALL       PGM(PGMC)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

If variables are used, the QRYSLT parameter would be typed as follows:

```
QRYSLT('AMT *EQ ' *CAT &VARAMT *CAT ' *OR  +
   LSTDAT *LE "' *CAT &VARDAT *CAT '"')
```

Note that the &VARAMT variable must be defined as a character type. If the variable is passed to your CL program as a numeric type, you must convert it to a character type to allow concatenation. You can use the Change Variable (CHGVAR) command to do this conversion.

Example 6: Using the Open Query File (OPNQRYF) command many times in a program

You can use the OPNQRYF command more than once in a high-level language program. For example, assume you want to prompt the user for some selection values, then display one or more pages of records. At the end of the first request for records, the user may want to specify other selection values and display those records. This can be done by doing the following:

1. Before calling the high-level language program, use an Override with Data Base File (OVRDBF) command to specify SHARE(*YES).
2. In the high-level language program, prompt the user for the selection values.
3. Pass the selection values to a CL program that issues the OPNQRYF command (or run the command with a call to program QCMDEXC). The file must be closed before your program processes the OPNQRYF command. You normally use the Close File (CLOF) command and monitor for the file not being open.
4. Return to the high-level language program.
5. Open the file in the high-level language program.
6. Process the records.
7. Close the file in the program.
8. Return to step 2.

When the program completes, run the Close File (CLOF) command or the Reclaim Resources (RCLRSC) command to close the file, then delete the Override with Data Base File command specified in step 1.

Note: An override command in a called CL program does not affect the open in the main program. All overrides are implicitly deleted when the program is ended. (However, you can use a call to program QCMDEXC from your high-level language program to specify an override, if needed.)

Example 7: Mapping fields for packed numeric data fields

Assume you have a packed decimal *Date* field in the format MMDDYY and you want to select all the records for the year 1988. You cannot select records directly from a

portion of a packed decimal field, but you can use the MAPFLD parameter on the OPNQRYF command to create a new field that you can then use for selecting part of the field.

The format of each mapped field definition is:

```
(result field  'expression'  attributes)
```

where:

result field	=	The name of the result field.
expression	=	How the result field should be derived. The expression can include substring, other built-in functions, or mathematical statements.
attributes	=	The optional attributes of the result field. If no attributes are given (or the field is not defined in a file) the OPNQRYF command calculates a field attribute determined by the fields in the expression.

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) QRYSLT('YEAR *EQ "88" ') +
             MAPFLD((CHAR6 '%DIGITS(DATE)')  +
               (YEAR '%SST(CHAR6 5 2)' *CHAR 2))
CALL       PGM(PGMC)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

The first mapped field definition specifies that the *Char6* field be created from the packed decimal *Date* field. The %DIGITS function converts from packed decimal to character and ignores any decimal definitions (that is, 1234.56 is converted to '123456'). Because no definition of the *Char6* field is specified, the system assigns a length of 6. The second mapped field defines the *Year* field as type *CHAR (character) and length 2. The expression uses the substring function to map the last 2 characters of the *Char6* field into the *Year* field.

Note that the mapped field definitions are processed in the order in which they are specified. In this example, the *Date* field was converted to character and assigned to the *Char6* field. Then, the last two digits of the *Char6* field (the year) were assigned to the *Year* field. Any changes to this order would have produced an incorrect result.

Note: Mapped field definitions are always processed before the QRYSLT parameter is evaluated.

You could accomplish the same result by specifying the substring on the QRYSLT parameter and dropping one of the mapped field definitions as follows:

```
OPNQRYF    FILE(FILEA) +
             QRYSLT('%SST(CHAR6 5 2) *EQ "88" ') +
             MAPFLD((CHAR6 '%DIGITS(DATE)'))
```

Example 8: Using the "wild card" function

Assume you have a packed decimal date field in the format MMDDYY and you want to select the records for March 1988. To do this, you can specify:

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) +
             QRYSLT('%DIGITS(DATE) *EQ %WLDCRD("03__88")')
CALL       PGM(PGMC)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

Note that the only time the MAPFLD parameter is needed to define a data base field for the result of the %DIGITS function is when the result needs to be used with a function that only supports a simple field name (not a function or expression) as an argument. The %WLDCRD operation has no such restriction on the operand that appears before the *EQ operator.

Note that although the field in the data base is in numeric form, double apostrophes surround the literal to make its definition the same as the *Char6* field.

The %WLDCRD function lets you select any records that match your selection values, in which the underline (_) will match any single character value. The two underline characters in Example 8 allow any day in the month of March to be selected. The %WLDCRD function also allows you to name the wild card character (underline is the default).

The wild card function supports two different forms:

- A fixed-position wild card as shown in the previous example in which the underline (or your designated character) matches any single character as in the following example:

  ```
  QRYSLT('FLDA *EQ %WLDCRD("A_C")')
  ```

  This compares successfully to ABC, ACC, ADC, AXC, and so on. In this example, the field being analyzed only compares correctly if it is exactly 3 characters in length. If the field is longer than 3 characters, you also need the second form of wild card support.
- A variable-position wild card will match any zero or more characters. The Open Query File (OPNQRYF) command uses an asterisk (*) for this type of wild card variable character or you can specify your own character. An asterisk is used in the following example:

  ```
  QRYSLT('FLDB *EQ %WLDCRD("A*C*") ')
  ```

  This compares successfully to AC, ABC, AXC, ABCD, AXXXXXXXC, and so on. The asterisk causes the command to ignore any intervening characters if they exist. Notice that in this example the asterisk is specified both before and after the character or characters that may appear later in the field. If the asterisk were omitted from the end of the search argument, it causes a selection only if the field ends with the character C.

  You must specify an asterisk at the start of the wild card string if you want to select records where the remainder of the pattern starts anywhere in the field. Similarly, the pattern string must end with an asterisk if you want to select records where the remainder of the pattern ends anywhere in the field.

  For example, you can specify:

  ```
  QRYSLT('FLDB *EQ %WLDCRD("*ABC*DEF*") ')
  ```

  You get a match on ABCDEF, ABCXDEF, ABCXDEFX, ABCXXXXXXDEF, ABCXXXDEFXXX, XABCDEF, XABCXDEFX, and so on.

You can combine the two wild card functions as in the following example:

```
QRYSLT('FLDB *EQ %WLDCRD("ABC_*DEF*") ')
```

You get a match on ABCxDEF, ABCxxxxxxDEF, ABCxxxDEFxxx, and so on. The underline forces at least one character to appear between the ABC and DEF (for example, ABCDEF would not match).

Assume you have a *Name* field that contains:

```
JOHNS
JOHNS SMITH
JOHNSON
JOHNSTON
```

If you specify the following you will only get the first record:

```
QRYSLT('NAME *EQ "JOHNS"')
```

You would not select the other records because a comparison is made with blanks added to the value you specified. The way to select all four names is to specify:

```
QRYSLT('NAME *EQ %WLDCRD("JOHNS*")')
```

Note: For information about using the %WLDCRD function for DBCS, see Appendix B.

Example 9: Using complex selection statements

Complex selection statements can also be specified. For example, you can specify:

```
QRYSLT('DATE *EQ "880101" *AND AMT *GT 5000.00')

QRYSLT('DATE *EQ "880101" *OR AMT *GT 5000.00')
```

You can also specify:

```
QRYSLT('CODE *EQ "A" *AND TYPE *EQ "X" *OR CODE *EQ "B")
```

The rules governing the priority of processing the operators are described in the *CL Reference*. Some of the rules are:

- The *AND operations are processed first; therefore, the record would be selected if:

  The *Code* field = "A" and The *Type* field = "X"
  or
  The *Code* field = "B"

- Parentheses can be used to control how the expression is handled, as in the following example:

```
QRYSLT('(CODE *EQ "A" *OR CODE *EQ "B") *AND TYPE *EQ "X" +
        *OR CODE *EQ "C"')
```

  The *Code* field = "A" and The *Type* field = "X"
  or
  The *Code* field = "B" and The *Type* field = "X"
  or
  The *Code* field = "C"

You can also use the symbols described in the *CL Reference* instead of the abbreviated form (for example, you can use = instead of *EQ) as in the following example:

```
QRYSLT('CODE = "A" & TYPE = "X" | AMT > 5000.00')
```

This command selects all records in which:

The *Code* field = "A" and The *Type* field = "X"
or
The *Amt* field > 5000.00

A complex selection statement can also be written, as in the following example:

```
QRYSLT('CUSNBR = %RANGE("60000" "69999") & TYPE = "B" +
       & SALES>0 & ACCRCV / SALES>.3')
```

This command selects all records in which:

The *Cusnbr* field is in the range 60000-69999 and
The *Type* field = "B" and
The *Sales* fields are greater than 0 and
Accrcv divided by *Sales* is greater than 30 percent

Specifying a Keyed Access Path without Using DDS

The dynamic access path function allows you to specify a keyed access path for the data to be processed. If an access path already exists that can be shared, the system may share it. If a new access path is required, it is built before any records are passed to the program.

Example 1: Arranging records using one key field

Assume you want to process the records in FILEA arranged by the value in the *Cust* field with program PGMD. You can specify:

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) KEYFLD(CUST)
CALL       PGM(PGMD)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

Note: The FORMAT parameter on the Open Query File (OPNQRYF) command is not needed because PGMD is created by specifying FILEA as the processed file. FILEA may be an arrival sequence or a keyed sequence file. If FILEA is keyed, its key field may be the *Cust* field or a totally different field.

Example 2: Arranging records using multiple key fields

If you want the records to be processed by *Cust* sequence and then by *Date* in *Cust*, specify:

```
OPNQRYF    FILE(FILEA) KEYFLD(CUST DATE)
```

If you want the *Date* to appear in descending sequence, specify:

```
OPNQRYF    FILE(FILEA) KEYFLD((CUST) (DATE *DESCEND))
```

In these two examples, the FORMAT parameter is not used. (If a different format is defined, all key fields must exist in the format.)

Specifying Key Fields from Different Files

A dynamic keyed access path over a join logical file allows you to specify a processing sequence in which the keys can be in different physical files (DDS restricts the keys to the primary file).

The specification is identical to the previous method. The access path is specified using whatever key fields are required. There is no restriction on which physical file the key fields are in. However, if a key field exists in other than the primary file of a join specification, the system must make a temporary copy of the joined records. The system must also build a keyed access path over the copied records before the query file is opened. The key fields must exist in the format identified on the FORMAT parameter.

Example 1: Using a field in a secondary file as a key field

Assume you already have a join logical file named JOINLF. FILEX is specified as the primary file and is joined to FILEY. You want to process the records in JOINLF by the *Descrp* field which is in FILEY.

Assume the file record formats contain the following fields:

FILEX	FILEY	JOINLF
Item	Item	Item
Qty	Descrp	Qty
		Descrp

You can specify:

```
OVRDBF    FILE(JOINLF) SHARE(*YES)
OPNQRYF   FILE(JOINLF) KEYFLD(DESCRP)
CALL      PGM(PGMC)
CLOF      OPNID(JOINLF)
DLTOVR    FILE(JOINLF)
```

If you want to arrange the records by *Qty* in *Descrp* (*Descrp* is the primary key field and *Qty* is a secondary key field) you can specify:

```
OPNQRYF   FILE(JOINLF) KEYFLD(DESCRP QTY)
```

Dynamically Joining Data Base Files without DDS

The dynamic join function allows you to join files without having to first specify DDS and create a join logical file. You must use the FORMAT parameter on the Open Query File (OPNQRYF) command to specify the record format for the join. You can join any physical or logical file including a join logical file and a view (DDS does not allow you to join logical files). You can specify either a keyed or arrival sequence access path. If keys are specified, they can be from any of the files included in the join (DDS restricts keys to just the primary file).

In the following examples, it is assumed that the file specified on the FORMAT parameter was created. You will normally want to create the file before you create the processing program so you can use the externally described data definitions.

The default for the join order (JORDER) keyword is used in all of the following examples. The default for the JORDER keyword is *ANY, which tells the system that

it can determine the order in which to join the files. That is, the system determines which file to use as the primary file and which as the secondary files. This allows the system to try to improve the performance of the join function.

Example 1: Dynamically joining files

Assume you want to join FILEA and FILEB. Assume the files contain the following fields:

FILEA	FILEB	JOINAB
Cust	Cust	Cust
Name	Amt	Name
Addr		Amt

The join field is *Cust* which exists in both files. Any record format name can be specified in the Open Query File (OPNQRYF) command for the join file. The file does not need a member. The records are not required to be in keyed sequence.

You can specify:

```
OVRDBF     FILE(JOINAB) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA FILEB) FORMAT(JOINAB) +
             JFLD((FILEA/CUST FILEB/CUST)) +
             MAPFLD((CUST 'FILEA/CUST'))
CALL       PGM(PGME) /* Created using file JOINAB as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(JOINAB)
```

File JOINAB is a physical file with no data. This is the file that contains the record format to be specified on the FORMAT keyword in the Open Query File (OPNQRYF) command.

Notice that the TOFILE parameter on the Override with Data Base File (OVRDBF) command specifies the name of the primary file for the join operation (the first file specified for the FILE parameter on the OPNQRYF command). In this example, the FILE parameter on the Open Query File (OPNQRYF) command identifies the files in the sequence they are to be joined (A to B). The format for the file is in the file JOINAB.

The JFLD parameter identifies the *Cust* field in FILEA to join to the *Cust* field in FILEB. Because the *Cust* field is not unique across all of the joined record formats, it must be qualified on the JFLD parameter. The system attempts to determine, in some cases, the most efficient values even if you do not specify the JFLD parameter on the Open Query File (OPNQRYF) command. For example, using the previous example, if you specified:

```
OPNQRYF    FILE(FILEA FILEB) FORMAT(JOINAB) +
             QRYSLT('FILEA/CUST *EQ FILEB/CUST') +
             MAPFLD((CUST 'FILEA/CUST'))
```

The system joins FILEA and FILEB using the *Cust* field because of the values specified for the QRYSLT parameter. Notice that in this example the JFLD parameter is not specified on the command. However, if either JDFTVAL(*ONLYDFT) or JDFTVAL(*YES) is specified on the OPNQRYF command, the JFLD parameter must be specified.

The MAPFLD parameter is needed on the Open Query File (OPNQRYF) command to describe which file should be used for the data for the *Cust* field in the record format for file JOINAB. If a field is defined on the MAPFLD parameter, its unqualified name (the *Cust* field in this case without the file name identification) can be used anywhere else in the OPNQRYF command. Because the *Cust* field is defined on the MAPFLD parameter, the first value of the JFLD parameter need not be qualified. For example, the same result could be achieved by specifying:

```
JFLD((CUST FILEB/CUST)) +
MAPFLD((CUST 'FILEA/CUST'))
```

Any other uses of the same field name in the Open Query File (OPNQRYF) command to indicate a field from a file other than the file defined by the MAPFLD parameter must be qualified with a file name.

Because no KEYFLD parameter is specified, the records appear in any sequence depending on how the Open Query File (OPNQRYF) command selects the records. You can force the system to arrange the records the same as the primary file. To do this, specify *FILE on the KEYFLD parameter. You can specify this even if the primary file is in arrival sequence.

The JDFTVAL parameter (similar to the JDFTVAL keyword in DDS) can also be specified on the Open Query File (OPNQRYF) command to describe what the system should do if one of the records is missing from the secondary file. In this example, the JDFTVAL parameter was not specified, so only the records that exist in both files are selected.

If you tell the system to optimize the results of the query (through parameters on the OPNQRYF command), it will generally try to use the file with the smallest number of records as the primary file. However, the system will also try to avoid building a temporary file. Depending on the selection and sequence values you specify, the file with the largest number of records could be the primary file.

You can force the system to follow the sequence of the join as you have specified it in the JFILE parameter on the Open Query File (OPNQRYF) command by specifying JORDER(*FILE). If JDFTVAL(*YES) or JDFTVAL(*ONLYDFT) is specified, the system will never change the join file sequence because a different sequence could cause different results.

Example 2: Reading only those records with secondary file records

Assume you want to join files FILEAB, FILECD, and FILEEF to select only those records with matching records in secondary files. Define a file JOINF and describe the format that should be used. Assume the record formats for the files contain the following fields:

FILEAB	FILECD	FILEEF	JOINF
Abitm	Cditm	Efitm	Abitm
Abord	Cddscp	Efcolr	Abord
Abdat	Cdcolr	Efqty	Cddscp
			Cdcolr
			Efqty

In this case, all field names in the files that make up the join file begin with a 2-character prefix (identical for all fields in the file) and end with a suffix that is identical across all the files (for example, *xxitm*). This makes all field names unique and avoids having to qualify them.

The *xxitm* field allows the join from FILEAB to FILECD. The two fields *xxitm* and *xxcolr* allow the join from FILECD to FILEEF. A keyed access path does not have to exist for these files. However, if a keyed access path does exist, performance can improve significantly because the system will attempt to use the existing access path to arrange and select records, where it can. If access paths do not exist, the system automatically creates and maintains them as long as the file is open.

```
OVRDBF     FILE(JOINF) TOFILE(FILEAB) SHARE(*YES)
OPNQRYF    FILE(FILEAB FILECD FILEEF) +
             FORMAT(JOINF) +
             JFLD((ABITM CDITM)(CDITM EFITM) +
             (CDCOLR EFCOLR))
CALL       PGM(PGME) /* Created using file JOINF as input */
CLOF       OPNID(FILEAB)
DLTOVR     FILE(JOINF)
```

The join field pairs do not have to be specified in the order shown above. For example, the same result is achieved with a JFLD parameter value of:

```
JFLD((CDCOLR EFCOLR)(ABITM CDITM) (CDITM EFITM))
```

The attributes of each pair of join fields do not have to be identical. Normal padding of character fields and decimal alignment for numeric fields occurs automatically.

The JDFTVAL keyword is not specified so *NO is assumed and no default values are used to construct join records. If you specified JDFTVAL(*YES) and there is no record in file FILECD that has the same join field value as a record in file FILEAB, defaults are used for the *Cddscp* and *Cdcolr* fields to join to file FILEEF. Using these defaults, a matching record may be found in file FILEEF (depending on if the default value matches a record in the secondary file). If not, a default value appears for these files and for the *Efqty* field.

Example 3: Using mapped fields as join fields

You can use fields defined on the MAPFLD parameter for either one of the join field pairs. This is useful when the key in the secondary file is defined as a single field

(for example, a 6-character date field) and there are separate fields for the same information (for example, month, day, and year) in the primary file. Assume FILEA has character fields *Year*, *Month*, and *Day* and needs to be joined to FILEB which has the *Date* field in YYMMDD format. Assume you have defined file JOINAB with the desired format. You can specify:

```
OVRDBF     FILE(JOINAB) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA FILEB) FORMAT(JOINAB) +
             JFLD((YYMMDD FILEB/DATE)) +
             MAPFLD((YYMMDD 'YEAR *CAT MONTH *CAT DAY'))
CALL       PGM(PGME) /* Created using file JOINAB as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(JOINAB)
```

The MAPFLD parameter defines the *YYMMDD* field as the concatenation of several fields from FILEA. You do not need to specify field attributes (for example, length or type) for the *YYMMDD* field on the MAPFLD parameter because the system calculates the attributes from the expression.

Handling Missing Records in Secondary Join Files

The system allows you to control whether to allow defaults for missing records in secondary files (similar to the JDFTVAL DDS keyword for a join logical file). You can also specify that only records with defaults be processed. This allows you to select only those records in which there is a missing record in the secondary file.

Example 1: Reading records from the primary file that do not have a record in the secondary file

In Example 1 under "Dynamically Joining Data Base Files without DDS" on page 9-16, the JDFTVAL parameter is not specified, so the only records read are the result of a successful join from FILEA to FILEB. If you want a list of the records in FILEA that do not have a match in FILEB, you can specify *ONLYDFT on the JDFTVAL parameter as shown in the following example:

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA FILEB) FORMAT(FILEA) +
             JFLD((CUST FILEB/CUST)) +
             MAPFLD((CUST 'FILEA/CUST')) +
             JDFTVAL(*ONLYDFT)
CALL       PGM(PGME) /* Created using file FILEA as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

JDFTVAL(*ONLYDFT) causes a record to be returned to the program only when there is no equivalent record in the secondary file (FILEB).

Because any values returned by the join operation for the fields in FILEB are defaults, it is normal to use only the format for FILEA. The records that appear are those that do not have a match in FILEB. The FORMAT parameter is required whenever the FILE parameter describes more than a single file, but the file name specified can be one of the files specified on the FILE parameter. The program is created using FILEA.

Conversely, you can also get a list of all the records where there is a record in FILEB that does not have a match in FILEA. You can do this by making the secondary file the primary file in all the specifications. You would specify:

```
OVRDBF     FILE(FILEB) SHARE(*YES)
OPNQRYF    FILE(FILEB FILEA) FORMAT(FILEB) JFLD((CUST FILEA/CUST)) +
             MAPFLD((CUST 'FILEB/CUST')) JDFTVAL(*ONLYDFT)
CALL       PGM(PGMF) /* Created using file FILEB as input */
CLOF       OPNID(FILEB)
DLTOVR     FILE(FILEB)
```

Note: The Override with Data Base File (OVRDBF) command in this example uses FILE(FILEB) because it must specify the first file on the OPNQRYF FILE parameter. The Close File (CLOF) command also names FILEB. The JFLD and MAPFLD parameters also changed. The program is created using FILEB.

Unique-Key Processing

Unique-key processing allows you to process only the first record of a group. The group is defined by one or more records with the same set of key values. Processing the first record implies that the records you receive will have unique keys.

When you use unique-key processing, you can only read the file sequentially.

Example 1: Reading only unique-key records

Assume you want to process FILEA, which has records with duplicate keys for the *Cust* field. You want only the first record for each unique value of the *Cust* field to be processed by program PGMF. You can specify:

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) KEYFLD(CUST) UNIQUEKEY(*ALL)
CALL       PGM(PGMF)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

Example 2: Reading records using only some of the key fields

Assume you want to process the same file with the sequence: *Slsman, Cust, Date*, but you want only one record per *Slsman* and *Cust*. Assume the records in the file are:

Slsman	Cust	Date	Record #
01	5000	880109	1
01	5000	880115	2
01	4025	880103	3
01	4025	880101	4
02	3000	880101	5

You specify the number of key fields that are unique, starting with the first key field.

```
OVRDBF     FILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) KEYFLD(SLSMAN CUST DATE) UNIQUEKEY(2)
CALL       PGM(PGMD)
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEA)
```

The following records are retrieved by the program:

Slsman	Cust	Date	Record #
01	4025	880101	4
01	5000	880109	1
02	3000	880101	5

Defining Fields Derived from Existing Field Definitions

Mapped field definitions:

- Allow you to create internal fields that specify selection values (see Example 7 under "Selecting Records without Using DDS" on page 9-8 for more information).
- Allow you to avoid confusion when the same field name occurs in multiple files (see Example 1 under "Dynamically Joining Data Base Files without DDS" on page 9-16 for more information).
- Allow you to create fields that exist only in the format to be processed, but not in the data base itself. This allows you to perform translate, substring, concatenation, and complex mathematical operations. The following examples describe this function.

Example 1: Using derived fields

Assume you have the *Price* and *Qty* fields in the record format. You can multiply one field by the other by using the Open Query File (OPNQRYF) command to create the derived *Exten* field. You want FILEA to be processed, and you have already created FILEAA. Assume the record formats for the files contain the following fields:

FILEA	FILEAA
Order	Order
Item	Item
Qty	Exten
Price	Brfdsc
Descrp	

The *Exten* field is a mapped field. Its value is determined by multiplying *Qty* times *Price*. It is not necessary to have either the *Qty* or *Price* field in the new format, but they can exist in that format, too if you wish. The *Brfdsc* field is a brief description of the *Descrp* field (it uses the first 10 characters).

Assume you have specified PGMF to process the new format. To create this program, use FILEAA as the file to read. You can specify:

```
OVRDBF     FILE(FILEAA) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FILEAA) +
             MAPFLD((EXTEN 'PRICE * QTY') +
             (BRFDSC 'DESCRP'))
CALL       PGM(PGMF) /* Created using file FILEAA as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEAA)
```

Notice that the attributes of the *Exten* field are those defined in the record format for FILEAA. If the value calculated for the field is too large, an exception is sent to the program.

It is not necessary to use the substring function to map to the *Brfdsc* field if you only want the characters from the beginning of the field. The length of the *Brfdsc* field is defined in the FILEAA record format.

All fields in the format specified on the FORMAT parameter must be described on the OPNQRYF command. That is, all fields in the output format must either exist in one of the record formats for the files specified on the FILE parameter or be defined on the MAPFLD parameter. If you have fields in the format on the FORMAT parameter that your program does not use, you can use the MAPFLD parameter to place zeros or blanks in the fields. Assume the *Fldc* field is a character field and the *Fldn* field is a numeric field in the output format, and you are using neither value in your program. You can avoid an error on the OPNQRYF command by specifying:

```
MAPFLD((FLDC ' " " ')(FLDN 0))
```

Notice quotation marks enclose a blank value. By using a constant for the definition of an unused field, you avoid having to create a unique format for each use of the OPNQRYF command.

Example 2: Using built-in functions

Assume you want to calculate a mathematical function that is the sine of the *Fldm* field in FILEA. First create a file (assume it is called FILEAA) with a record format containing the following fields:

FILEA	FILEAA
Code	Code
Fldm	Fldm
	Sinm

You can then create a program (assume PGMF) using FILEAA as input and specify:

```
OVRDBF     FILE(FILEAA) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FILEAA) +
               MAPFLD((SINM '%SIN(FLDM)'))
CALL       PGM(PGMF) /* Created using file FILEAA as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEAA)
```

The built-in function %SIN calculates the sine of the field specified as its argument. Because the *Sinm* field is defined in the format specified on the FORMAT parameter, the OPNQRYF command converts its internal definition of the sine value (in floating point) to the definition of the *Sinm* field. This technique can be used to avoid certain high-level language restrictions regarding the use of floating-point fields. For example, if you defined the *Sinm* field as a packed decimal field, PGMF could be written using any high-level language, even though the value was built using a floating-point field.

There are many other functions besides sine that can be used. Refer to the OPNQRYF command in the *CL Reference* for a complete list of built-in functions.

Example 3: Using derived fields and built-in functions

Assume, in the previous example, that a field called *Fldx* also exists in FILEA, and the *Fldx* field has appropriate attributes used to hold the sine of the *Fldm* field. Also assume that you are not using the contents of the *Fldx* field. You can use the MAPFLD parameter to change the contents of a field before passing it to your high-level language program. For example, you can specify:

```
OVRDBF      FILE(FILEA) SHARE(*YES)
OPNQRYF     FILE(FILEA) MAPFLD((FLDX '%SIN(FLDM)'))
CALL        PGM(PGMF) /* Created using file FILEA as input */
CLOF        OPNID(FILEA)
DLTOVR      FILE(FILEA)
```

In this case, you do not need to specify a different record format on the FORMAT parameter. (The default uses the format of the first file on the FILE parameter.) Therefore, the program is created by using FILEA. When using this technique, you must ensure that the field you redefine has attributes that allow the calculated value to process correctly. The least complicated approach is to create a separate file with the specific fields you want to process for each query.

You can also use this technique with a mapped field definition and the %XLATE function to translate a field so that it appears to the program in a different manner than what exists in the data base. For example, you can translate a lowercase field so the program only sees uppercase.

The example described uses FILEA as an input file. You can also update data using the OPNQRYF command. However, if you use a mapped field definition to change a field, updates to the field are ignored.

Handling Divide by Zero

Dividing by zero is considered an error by the Open Query File (OPNQRYF) command.

Record selection is normally done before field mapping errors occur (for example, where field mapping would cause a division error). Therefore, a record can be omitted (based on the QRYSLT parameter values and valid data in the record) that would have caused a divide-by-zero error. In such an instance, the record would be omitted and processing by the OPNQRYF command would continue.

If you want a zero answer, the following describes a solution that is practical for typical commercial data.

Assume you want to divide A by B giving C (stated as A / B = C). Assume the following definitions where B may be zero.

Field	Digits	Dec
A	6	2
B	3	0
C	6	2

The following algorithm can be used:

```
(A * B) / %MAX((B * B) .nnnn1)
```

The %MAX function returns the maximum value of either B * B or a small value. The small value must have enough leading zeros so that it is less than any value calculated by B * B unless B is zero. In this example, B has zero decimal positions so .1 could be used. The number of leading zeros should be 2 times the number of decimals in B. For example, if B had 2 decimal positions, then .00001 should be used.

Specify the following MAPFLD definition:

```
MAPFLD((C '(A * B) / %MAX((B * B) .1)'))
```

The intent of the first multiplication is to produce a zero dividend if B is zero. This will ensure a zero result when the division occurs. Dividing by zero does not occur if B is zero because the .1 value will be the value used as the divisor.

Summarizing Data from Data Base File Records (Grouping)

The group processing function allows you to summarize data from existing data base records. You can specify:

- The grouping fields
- Selection values both before and after grouping
- A keyed access path over the new records
- Mapped field definitions that allow you to do such functions as sum, average, standard deviation, and variance, as well as counting the records in each group

You normally start by creating a file with a record format containing only the following types of fields:

- Grouping fields. Specified on the GRPFLD parameter that define groups. Each group contains a constant set of values for all grouping fields. The grouping fields do not need to appear in the record format identified on the FORMAT parameter.
- Aggregate fields. Defined by using the MAPFLD parameter with one or more of the following built-in functions:

%COUNT	Counts the records in a group
%SUM	A sum of the values of a field over the group
%AVG	Arithmetic average (mean) of a field, over the group
%MAX	Maximum value in the group for the field
%MIN	Minimum value in the group for the field
%STDDEV	Standard deviation of a field, over the group
%VAR	Variance of a field, over the group

- Constant fields. Allow constants to be placed in field values. The restriction that the Open Query File (OPNQRYF) command must know all fields in the output format is also true for the grouping function.

When you use group processing, you may only read the file sequentially.

Example 1: Using group processing

Assume you want to group the data by customer number and analyze the amount field. Your data base file is FILEA and you create a file named FILEAA containing a record format with the following fields:

FILEA	FILEAA
Cust	Cust
Type	Count (count of records per customer)
Amt	Amtsum (summation of the amount field)
	Amtavg (average of the amount field)
	Amtmax (maximum value of the amount field)

When you define the fields in the new file, you must ensure that they are large enough to hold the results. For example, if the *Amt* field is defined as 5 digits, you may want to define the *Amtsum* field as 7 digits. Any arithmetic overflow causes your program to end abnormally.

Assume the records in FILEA have the following values:

Cust	Type	Amt
001	A	500.00
001	B	700.00
004	A	100.00
002	A	1200.00
003	B	900.00
001	A	300.00
004	A	300.00
003	B	600.00

You then create a program (PGMG) using FILEAA as input to print the records.

```
OVRDBF     FILE(FILEAA) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FILEAA) KEYFLD(CUST) +
             GRPFLD(CUST) MAPFLD((COUNT '%COUNT') +
               (AMTSUM '%SUM(AMT)') +
               (AMTAVG '%AVG(AMT)') +
               (AMTMAX '%MAX(AMT)'))
CALL       PGM(PGMG) /* Created using file FILEAA as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FILEAA)
```

The records retrieved by the program appear as:

Cust	Count	Amtsum	Amtavg	Amtmax
001	3	1500.00	500.00	700.00
002	1	1200.00	1200.00	1200.00
003	2	1500.00	750.00	900.00
004	2	400.00	200.00	300.00

Note: If you specify the GRPFLD parameter, the groups may not appear in ascending sequence. To ensure a specific sequence, you should specify the KEYFLD parameter.

Assume you want to print only the summary records in this example in which the *Amtsum* value is greater than 700.00. Because the *Amtsum* field is an aggregate field for a given customer, use the GRPSLT parameter to specify selection after grouping. Add the GRPSLT parameter:

```
GRPSLT('AMTSUM *GT 700.00') +
```

The records retrieved by your program are:

Cust	Count	Amtsum	Amtavg	Amtmax
001	3	1500.00	500.00	700.00
002	1	1200.00	1200.00	1200.00
003	2	1500.00	750.00	900.00

The Open Query File (OPNQRYF) command supports selection both before grouping (QRYSLT parameter) and after grouping (GRPSLT parameter).

Assume you want to select additional customer records in which the *Type* field is equal to A. Because *Type* is a field in the record format for file FILEA and not an aggregate field, you add the QRYSLT statement to select before grouping as follows:

```
QRYSLT('TYPE *EQ "A" ')
```

Note that fields used for selection do not have to appear in the format processed by the program.

The records retrieved by your program are:

Cust	Count	Amtsum	Amtavg	Amtmax
001	2	800.00	400.00	500.00
002	1	1200.00	1200.00	1200.00

Notice the values for CUST 001 changed because the selection took place before the grouping took place.

Assume you want to arrange the output by the *Amtavg* field in descending sequence, in addition to the previous QRYSLT parameter value. You can do this by changing the KEYFLD parameter on the OPNQRYF command as:

```
KEYFLD((AMTAVG *DESCEND))
```

The records retrieved by your program are:

Cust	Count	Amtsum	Amtavg	Amtmax
002	1	1200.00	1200.00	1200.00
001	2	800.00	400.00	500.00

Final Total-Only Processing

Final-total-only processing is a special form of grouping in which you do not specify grouping fields. Only one record is output. All of the special built-in functions for grouping can be specified. You can also specify the selection of records that make up the final total.

Example 1: Simple total processing

Assume you have a data base file FILEA and decide to create file FINTOT for your final total record as follows:

FILEA	FINTOT
Code	Count (count of all the selected records)
Amt	Totamt (total of the amount field)
	Maxamt (maximum value in the amount field)

The FINTOT file is created specifically to hold the single record which is created with the final totals. You would specify:

```
OVRDBF     FILE(FINTOT) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FINTOT) +
             MAPFLD((COUNT '%COUNT') +
               (TOTAMT '%SUM(AMT)') (MAXAMT '%MAX(AMT)'))
CALL       PGM(PGMG) /* Created using file FINTOT as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FINTOT)
```

Example 2: Total-only processing with record selection

Assume you want to change the previous example so that only the records where the *Code* field is equal to B are in the final total. You can add the QRYSLT parameter as follows:

```
OVRDBF     FILE(FINTOT) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FINTOT) +
             QRYSLT('CODE *EQ "B" ') MAPFLD((COUNT '%COUNT') +
               (TOTAMT '%SUM(AMT)') (MAXAMT '%MAX(AMT)'))
CALL       PGM(PGMG) /* Created using file FINTOT as input */
CLOF       OPNID(FILEA)
DLTOVR     FILE(FINTOT)
```

You can use the GRPSLT keyword with the final total function. The GRPSLT selection values you specify determines if you receive the final total record.

Example 3: Total-only processing using a new record format

Assume you want to process the new file/format with a CL program. You want to read the file and send a message with the final totals. You can specify:

```
DCLF       FILE(FINTOT)
DCL        &COUNTA *CHAR LEN(7)
DCL        &TOTAMA *CHAR LEN(9)
OVRDBF     FILE(FINTOT) TOFILE(FILEA) SHARE(*YES)
OPNQRYF    FILE(FILEA) FORMAT(FINTOT) MAPFLD((COUNT '%COUNT') +
                (TOTAMT '%SUM(AMT)'))
RCVF
CLOF       OPNID(FILEA)
CHGVAR     &COUNTA &COUNT
CHGVAR     &TOTAMTA &TOTAMT
SNDPGMMSG  MSG('COUNT=' *CAT &COUNTA *CAT +
               ' Total amount=' *CAT &TOTAMTA)
DLTOVR     FILE(FINTOT)
```

You must convert the numeric fields to character fields to include them in an impromptu message.

Controlling How the System Runs the Open Query File Command

The optimization function allows you to specify how you are going to use the results of the query.

When you use the Open Query File (OPNQRYF) command there are two steps where performance considerations exist. The first step is during the actual processing of the OPNQRYF command itself. This step decides if OPNQRYF is going to use an existing access path or build a new one for this query request. The second step when performance considerations play a role is when the application program is using the results of the OPNQRYF to process the data.

For most batch type functions, you are usually only interested in the total time of both steps mentioned above. Therefore, the default for OPNQRYF is OPTIMIZE(*ALLIO). This means that OPNQRYF will consider the total time it takes for both steps.

If you use OPNQRYF in an interactive environment, you may not be interested in processing the entire file. You may want the first screen full of records to be displayed as quickly as possible. For this reason, you would want the first step to avoid building an access path, if possible. You might specify OPTIMIZE(*FIRSTIO) in such a situation.

If you want to process the same results of OPNQRYF with multiple programs, you would want the first step to make an efficient open data path (ODP). That is, you would try to minimize the number of records that must be read by the processing program in the second step by specifying OPTIMIZE(*MINWAIT) on the OPNQRYF command.

If the KEYFLD or GRPFLD parameters on the OPNQRYF command require that an access path be built when there is no access path to share, the access path is built entirely regardless of the OPTIMIZE entry. Optimization mainly affects selection processing.

Example 1: Optimizing for the first set of records

Assume that you have an interactive job in which the operator requests all records where the *Code* field is equal to B. Your program's subfile contains 15 records per screen. You want to get the first screen of results to the operator as quickly as possible. You can specify:

```
OVRDBF    FILE(FILEA) SHARE(*YES)
OPNQRYF   FILE(FILEA) QRYSLT('CODE = "B" ') +
            SEQONLY(*YES 15) OPTIMIZE(*FIRSTIO)
CALL      PGM(PGMA)
CLOF      OPNID(FILEA)
DLTOVR    FILE(FILEA)
```

The system optimizes handling the query and fills the first buffer with records before completing the entire query regardless of whether an access path already exists over the *Code* field.

Example 2: Optimizing to minimize the number of records read

Assume that you have multiple programs that will access the same file which is built by the Open Query File (OPNQRYF) command. In this case, you will want to optimize the performance so that the application programs read only the data they are interested in. This means that you want OPNQRYF to perform the selection as efficiently as possible. You could specify:

```
OVRDBF    FILE(FILEA) SHARE(*YES)
OPNQRYF   FILE(FILEA) QRYSLT('CODE *EQ "B"') +
          KEYFLD(CUST) OPTIMIZE(*MINWAIT)
CALL      PGM(PGMA)
POSDBF    OPNID(FILEA) POSITION(*START)
CALL      PGM(PGMB)
CLOF      OPNID(FILEA)
DLTOVR    FILE(FILEA)
```

Considerations for Creating a File and Using the FORMAT Parameter

You must specify a record format name on the FORMAT parameter when you request join processing by specifying multiple entries on the FILE parameter (that is, you cannot specify FORMAT(*FILE)). Also, a record format name is normally specified with the grouping function or when you specify a complex expression on the MAPFLD parameter to define a derived field. Consider the following:

- The record format name is any name you select. It can differ from the format name in the data base file you want to query.
- The field names are any names you select. If the field names are unique in the data base files you are querying, the system implicitly maps the values for any fields with the same name in a queried file record format (FILE parameter) and in the query result format (FORMAT parameter). See Example 1 under "Dynamically Joining Data Base Files without DDS" on page 9-16 for more information.
- If the field names are unique, but the attributes differ between the file specified on the FILE parameter and the file specified on the FORMAT parameter, the data is implicitly mapped.
- The correct field attributes must be used when using the MAPFLD parameter to define derived fields. For example, if you are using the grouping %SUM function, you must define a field that is large enough to contain the total. If not, an arithmetic overflow occurs and an exception is sent to the program.

- Decimal alignment occurs for all field values mapped to the record format identified on the FORMAT parameter. Assume you have a field in the query result record format with 5 digits with 0 decimals, and the value that was calculated or must be mapped to that field is 0.12345. You will receive a result of 0 in your field because digits to the right of the decimal point are truncated.

Considerations for Arranging Records

The default processing for the Open Query File (OPNQRYF) command provides records in any order that improves performance and does not conflict with the order specified on the KEYFLD parameter. Therefore, unless you specify the KEYFLD parameter to either name specific key fields or specify KEYFLD(*FILE), the sequence of the records returned to your program can vary each time you run the same Open Query File (OPNQRYF) command.

Considerations for DDM Files

The Open Query File (OPNQRYF) command can process DDM files. All files identified on the FILE parameter must exist on the same IBM AS/400 system or System/38 target system. An OPNQRYF which specifies group processing and uses a DDM file requires that both the source and target system be the same type (either both System/38 or both AS/400 systems).

Considerations for Writing a High-Level Language Program

For the method described under "Using an Existing Record Format in the File" on page 9-3 (where the FORMAT parameter is omitted), your high-level language program is coded as if you are directly accessing the data base file. Selection or sequencing occurs external to your program, and the program receives the selected records in the order you specified. The program does not receive records that are omitted by your selection values. This same function occurs if you process through a logical file with select/omit values.

If you use the FORMAT parameter, your program specifies the same file name used on the FORMAT parameter. The program is written as if this file contains actual data.

If you read the file sequentially, your high-level language may automatically specify that the key fields are ignored. Normally you write the program as if it is reading records in arrival sequence. If the KEYFLD parameter is used on the Open Query File (OPNQRYF) command, you receive a diagnostic message, which can be ignored.

If you process the file randomly by keys, your high-level language probably requires a key specification. If you have selection values, it may prevent your program from accessing a record that exists in the data base. A Record not found condition can occur on a random read whether the OPNQRYF command was used or whether a logical file created using DDS select/omit logic was used.

In some cases, you can monitor exceptions caused by mapping errors such as arithmetic overflow, but it is better to define the attributes of all fields to correctly handle the results.

Messages Sent When the Open Query File (OPNQRYF) Command Is Run

When the OPNQRYF command is run, messages are sent informing the interactive user of the status of the OPNQRYF request. For example, a message would be sent to the user if a keyed access path was built by the OPNQRYF to satisfy the request. The following messages might be sent during a run of the OPNQRYF command:

Message Identifier	Description
CPI4301	Query running.
CPI4302	Query running. Building access path...
CPI4303	Query running. Creating copy of file...
CPI4304	Query running. Selection complete...
CPI4305	Query running. Sorting copy of file...
CPI4306	Query running. Building access path from file...
CPI4311	Query running. Number of records processed...

To stop these status messages from appearing, see the discussion about message handling in the *CL Programmer's Guide*.

Using the Open Query File (OPNQRYF) Command for More Than Just Input

The OPNQRYF command supports the OPTION parameter to determine the type of processing. The default is OPTION(*INP), so the file is opened for input only. You can also use other OPTION values on the OPNQRYF command and a high-level language program to add, update, or delete records through the open query file. However, if you specify the UNIQUEKEY, GRPFLD, or GRPSLT parameters, use one of the aggregate functions, or specify multiple files on the FILE parameter, your use of the file is restricted to input only.

A join logical file is limited to input-only processing. A view is limited to input-only processing, if group, join, union, or distinct processing is specified in the definition of the view.

If you want to change a field value from the current value to a different value in some of the records in a file, you can use a combination of the OPNQRYF command and a specific high-level language program. For example, assume you want to change all the records where the *Flda* field is equal to ABC so that the *Flda* field is equal to XYZ. You can specify:

```
OVRDBF    FILE(FILEA) SHARE(*YES)
OPNQRYF   FILE(FILEA) OPTION(*ALL) QRYSLT('FLDA *EQ "ABC" ')
CALL      PGM(PGMA)
CLOF      OPNID(FILEA)
DLTOVR    FILE(FILEA)
```

Program PGMA processes all records it can read, but the query selection restricts these to records where the *Flda* field is equal to ABC. The program changes the field value in each record to XYZ and updates the record.

You can also delete records in a data base file using the OPNQRYF command. For example, assume you have a field in your record that, if equal to X, means the

record should be deleted. Your program can be written to delete any records it reads and use the OPNQRYF command to select those to be deleted such as:

```
OVRDBF    FILE(FILEA) SHARE(*YES)
OPNQRYF   FILE(FILEA) OPTION(*ALL) QRYSLT('DLTCOD *EQ "X" ')
CALL      PGM(PGMB)
CLOF      OPNID(FILEA)
DLTOVR    FILE(FILEA)
```

You can also add records by using the OPNQRYF command. However, if the query specifications include selection values, your program may be prevented from reading the added records because of the selection values.

Using the Open Query File (QPNQRYF) Command for Random Processing

Most of the previous examples show the OPNQRYF command using sequential processing. Random processing operations (for example, the RPG/400 operation CHAIN or the COBOL/400 operation READ) can be used in most cases. However, if you are using the group or unique-key functions, you cannot process the file randomly.

Performance Considerations

The best performance can occur when the Open Query File (OPNQRYF) command uses an existing keyed access path. For example, if you want to select all the records where the *Code* field is equal to B and an access path exists over the *Code* field, the system may use the access path to perform the selection rather than read the records and select at run time (dynamic selection).

Part of the OPNQRYF processing is to determine what is the fastest approach to satisfying your request. If the file you are using is large and most of the records have the *Code* field equal to B, it is faster to use arrival sequence processing than to use an existing keyed access path. Your program will still see the same records. OPNQRYF can only make this type of decision if an access path exists on the *Code* field. In general, if your request will include approximately 20% or more of the number of records in the file, OPNQRYF will tend to ignore the existing access paths and read the file in arrival sequence. You can force the use of an existing access path in most cases by specifying OPTIMIZE(*FIRSTIO) on the OPNQRYF command.

If no access path exists over the *Code* field, the program reads all of the records in the file and passes only the selected records to your program. That is, the file is processed in arrival sequence.

The system can perform selection faster than your application program. If no appropriate keyed access path exists, either your program or the system makes the selection of the records you want to process. Allowing the system to perform the selection process is considerably faster than passing all the records to your application program.

This is especially true if you are opening a file for update operations because individual records must be passed to your program, and locks are placed on every record read (in case your program needs to update the record). By letting the system perform the record selection, the only records passed to your program and locked are those that meet your selection values.

If you use the KEYFLD parameter to request a specific sequence for reading records, the fastest performance results if an access path already exists that uses the same key specification or if a keyed access path exists that is similar to your specifications

(such as a key that contains all the fields you specified plus some additional fields on the end of the key). This is also true for the GRPFLD parameter and on the to-fields of the JFLD parameter. If no such access path exists, the system builds an access path and maintains it as long as the file is open in your job.

Processing all of the records in a file by an access path that does not already exist is generally not as efficient as using a full record sort, if the number of records to be arranged (not necessarily the total number of records in the file) exceeds 1000. While it is generally faster to build the keyed access path than to do the sort, faster processing allowed by the use of arrival sequence processing normally favors sorting the data when looking at the total job time. If a usable access path already exists, using the access path can be faster than sorting the data.

If you use the grouping function, faster performance is achieved if you specify selection before grouping (QRYSLT parameter) instead of selection after grouping (GRPSLT parameter). Only use the GRPSLT parameter for comparisons involving aggregate functions.

For most uses of the OPNQRYF command, new or existing access paths are used to access the data and present it to your program. In some cases, the system must create a temporary file and then build an access path over the temporary structure. The rules for this situation are complex, but the following are typical cases in which this occurs:

- When you specify both the GRPFLD and KEYFLD parameters, but they are not the same
- When you specify a dynamic join, and the KEYFLD parameter describes key fields from different physical files

When a dynamic join occurs, OPNQRYF will attempt to improve performance by joining the file with the smallest number of records with the file with the largest number of records. OPNQRYF tries to avoid building a temporary file. Consequently, if your sequence specifications cause a temporary result, OPNQRYF may join the file with the large number of records to the file with the small number of records thus causing slower performance. You can force the join sequence by specifying JORDER(*FILE).

Performance Comparisons with Other Data Base Functions

The Open Query File (OPNQRYF) command uses the same data base support as logical files and join logical files. Therefore, the performance of functions like building a keyed access path or doing a join operation will be the same.

The selection functions done by the OPNQRYF command (for the QRYSLT and GRPSLT parameters) are similar to logical file select/omit. The main difference is that for the OPNQRYF command, the system decides whether to use access path selection or dynamic selection (similar to omitting or specifying the DYNSLT keyword in the DDS for a logical file), as a result of the access paths available on the system and what value was specified on the OPTIMIZE parameter.

Considerations for Field Use

When the grouping function is used, all fields in the record format for the open query file (FORMAT parameter) and all key fields (KEYFLD parameter) must either be grouping fields (specified on the GRPFLD parameter) or mapped fields (specified on the MAPFLD parameter) that are defined using only grouping fields, constants, and aggregate functions. The aggregate functions are: %AVG, %COUNT, %MAX (using only one operand), %MIN (using only one operand), %STDDEV, %SUM, and %VAR. Group processing is required in the following cases:

- When you specify grouping field names on the GRPFLD parameter
- When you specify group selection values on the GRPSLT parameter
- When a mapped field that you specified on the MAPFLD parameter uses an aggregate function in its definition

Fields contained in a record format, identified on the FILE parameter, and defined (in the DDS used to create the file) with a usage value of N (neither input nor output) cannot be specified on any parameter of the OPNQRYF command. Only fields defined as either I (input-only) or B (both input and output) usage can be specified. Any fields with usage defined as N in the record format identified on the FORMAT parameter are ignored by the OPNQRYF command.

Fields in the open query file records normally have the same usage attribute (input-only or both input and output) as the fields in the record format identified on the FORMAT parameter, with the exceptions noted below. If the file is opened for any option (OPTION parameter) that includes output or update and any usage, and if any B (both input and output) field in the record format identified on the FORMAT parameter is changed to I (input only) in the open query file record format, then an information message is sent by the OPNQRYF command.

If you request join processing or group processing, or if you specify UNIQUEKEY processing, all fields in the query records are given input-only use. Any mapping from an input-only field from the file being processed (identified on the FILE parameter) is given input-only use in the open query file record format. Fields defined using the MAPFLD parameter are normally given input-only use in the open query file. A field defined on the MAPFLD parameter is given a value that matches the use of its constituent field if all of the following are true:

- Input-only is not required because of any of the conditions previously described in this section.
- The field-definition expression specified on the MAPFLD parameter is a field name (no operators or built-in functions).
- The field used in the field-definition expression exists in one of the file, member, or record formats specified on the FILE parameter (not in another field defined using the MAPFLD parameter).
- The base field and the mapped field are compatible field types (the mapping does not mix numeric and character field types, unless the mapping is between zoned and character fields of the same length).
- If the base field is binary with nonzero decimal precision, the mapped field must also be binary and have the same precision.

Considerations for Files Shared in a Job

In order for your application program to use the open data path built by the Open Query File (OPNQRYF) command, your program must share the query file. If your program does not open the query file as shared, then it actually does a full open of the file it was originally compiled to use (not the query open data path built by the OPNQRYF command). Your program will share the query open data path, depending on the following conditions:

- Your application program must open the file as shared. Your program meets this condition when the first or only member queried (as specified on the FILE parameter) has an attribute of SHARE(*YES). If the first or only member has an attribute of SHARE(*NO), then you must specify SHARE(*YES) in an Override with Data Base File (OVRDBF) command before calling your program.
- The file opened by your application program must have the same name as the file opened by the OPNQRYF command. Your program meets this condition when the file specified in your program has the same file and member name as the first or only member queried (as specified on the FILE parameter). If the first or only member has a different name, then you must specify an Override with Data Base File (OVRDBF) command of the name of the file your program was compiled against to the name of the first or only member queried.

The OPNQRYF command never shares an existing open data path in the job. If an open data path that can be shared already exists in the job with the same file, library, and member name as the query open data path, an error message is sent to report the error and the query file is not opened.

Subsequent shared opens adhere to the same open options (such as SEQONLY) that were in effect when the OPNQRYF command was run.

See "Sharing Data Base Files in the Same Job" on page 8-7 for more information about sharing files in a job.

Considerations for Checking If the Record Format Description Changed

If record format level checking is indicated, the format level number of the open query file record format (identified on the FORMAT parameter) is checked against the record format your program was compiled against. This occurs when your program shares the previously opened query file. Your program's shared open is checked for record format level if the following conditions are met:

- The first or only file queried (as specified on the FILE parameter) must have the LVLCHK(*YES) attribute.
- There must not be an override of the first or only file queried to LVLCHK(*NO).

Other Run Time Considerations

Overrides can change the name of the file, library, and member that should be processed by the open query file. (However, any parameter values other than TOFILE, MBR, LVLCHK, INHWRT, or SEQONLY specified on an Override with Data Base File (OVRDBF) command are ignored by the OPNQRYF command.) If a name change override applies to the first or only member queried, any additional overrides must be against the new name, not the name specified for the FILE parameter on the OPNQRYF command.

Typical Errors When Using the Open Query File (OPNQRYF) Command

Several functions must be correctly specified for the OPNQRYF command and your program to get the correct results. The Display Job (DSPJOB) command is your most useful tool if problems occur. This command supports both the open files option and the file overrides option. You should look at both of these if you are having problems.

These are the most common problems and how to correct them:

- Shared open data path (ODP). The OPNQRYF command operates through a shared ODP. In order for the file to process correctly, the member must be opened for a shared ODP. If you are having problems, use the open files option on the DSPJOB command to determine if the member is opened and has a shared ODP.

  There are normally two reasons that the file is not open:
  - The member to be processed must be SHARE(*YES). Either use an Override with Data Base File (OVRDBF) command or permanently change the member.
  - The file is closed. If you operate from a command entry line and successfully run the OPNQRYF command, then run a command that fails and sends an error message, the command entry function internally runs a Reclaim Resources (RCLRSC) command. This closes all open files, including the open query file, unless TYPE(*PERM) was specified on the OPNQRYF command. You must run the OPNQRYF command again.
- Level check. Level checking is normally used because it ensures that your program is running against the same record format that the program was compiled with. If you are experiencing level check problems, it is normally because of one of the following:
  - The record format was changed since the program was created. Creating the program again should correct the problem.
  - An override is directing the program to an incorrect file. Use the file overrides option on the DSPJOB command to ensure that the overrides are correctly specified.
  - The FORMAT parameter is needed but is either not specified or incorrectly specified. When a file is processed with the FORMAT parameter, you must ensure:
    - The Override with Data Base File (OVRDBF) command, used with the TOFILE parameter, describes the first file on the FILE parameter of the Open Query File (OPNQRYF) command.
    - The FORMAT parameter identifies the file that contains the format used to create the program.
  - The FORMAT parameter is used to process a format from a different file (for example, for group processing), but SHARE(*YES) was not requested on the OVRDBF command.
- The file to be processed is at end of file. The normal use of the OPNQRYF command is to process a file sequentially where you can only process the file once. At that point, the position of the file is at the end of the file and you will not receive any records if you attempt to process it again. To process the file again from the start, you must either run the OPNQRYF command again or reposition the file before processing. You can reposition the file by using the Position Data Base File (POSDBF) command, or through a high-level language program statement.
- No records exist. This can be caused when you use the FORMAT keyword, but do not specify the OVRDBF command.

- Syntax errors. The system found an error in the specification of the OPNQRYF command.
- Operation not valid. The definition of the query does not include the KEYFLD parameter, but the high-level language program attempts to read the query file using a key field.
- Get option not valid. The high-level language program attempted to read a record or set a record position before the current record position, and the query file used either the group by option, the unique key option, or the distinct option on the SQL/400 statement.

Chapter 10. Basic Data Base File Operations

The basic data base file operations that can be performed in a program are discussed in this chapter. The operations include: setting a position in the data base file, reading records from the file, updating records in the file, adding records to the file, and deleting records from the file.

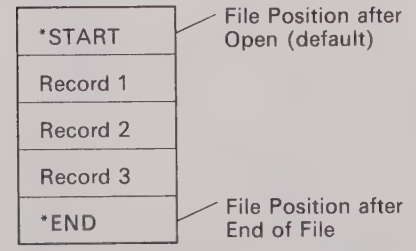
Setting a Position in the File

After a file is opened by a job, the system maintains a position in the file for that job. The file position is used in processing the file. For example, if a program does a read operation requesting the next sequential record, the system uses the file position to determine which record to return to the program. The system will then set the file position to the record just read, so that another read operation requesting the next sequential record will return the correct record. The system keeps track of all file positions for each job. In addition, each job can have multiple positions in the same file.

The file position is first set to the position specified in the POSITION parameter on the Override with Data Base File (OVRDBF) command. If you do not use an OVRDBF command, or if you take the default for the POSITION parameter, the file position is set just before the first record in the member's access path.

A program can change the current file position by using the appropriate high-level language program file positioning operation (for example, SETLL in RPG/400 or START in COBOL/400). A program can also change the file position by using the CL Position Data Base File (POSDBF) command.

At end of file, after the last read, the file member is positioned to *START or *END file positions, depending on whether the program was reading forward or backward through the file. The following diagram shows *START and *END file positions.



Only a read operation, force-end-of-data operation, high-level language positioning operation, or specific CL command to change the file position can change the file position. Add, update, and delete operations do not change the file position. After a read operation, the file is positioned to the new record. This record is then returned to your program. After the read operation is completed, the file is positioned at the record just returned to your program. If the member is open for input, a force-end-of-data operation positions the file after the last record in the file (*END) and sends the end-of-file message to your program.

For sequential read operations, the current file position is used to locate the next or previous record on the access path. For read-by-key or read-by-relative-record-number operations, the file position is not used. If POSITION(*NONE) is specified at open time, no starting file position is set. In this case, you must establish a file position in your program, if you are going to read sequentially.

If end-of-file delay was specified for the file on an Override With Data Base File (OVRDBF) command, the file is not positioned to *START or *END when the program reads the last record. The file remains positioned at the last record read. A file with end-of-file delay processing specified is positioned to *START or *END only when a force-end-of-data (FEOD) occurs or a controlled job end occurs. For more information about end-of-file delay, see "Waiting for More Records When End of File Is Reached" on page 10-4.

You can also use the Position Data Base File (POSDBF) command to set or change the current position in your file for files opened using either the Open Data Base File (OPNDBF) command or the Open Query File (OPNQRYF) command. For more information about the POSDBF command, see the *CL Reference*.

Reading Data Base Records

The AS/400 system provides a number of ways to read data base records. The next sections describe those ways in detail. (Some high-level languages do not support all of the read operations available on the system. See your high-level language guide for more information about reading data base records.)

Reading Data Base Records Using an Arrival Sequence Access Path

The system performs the following read operations based on the operations you specify using your high-level language. These operations are allowed if the file was defined with an arrival sequence access path; or if the file was defined with a keyed sequence access path with the ignore-keyed-sequence-access-path option specified in the program, on the Open Data Base File (OPNDBF) command, or on the Open Query File (OPNQRYF) command. See "Ignoring the Keyed Sequence Access Path" on page 8-3 for more details about the option to ignore a keyed sequence access path.

Note: Your high-level language may not allow all of the following read operations. Refer to your high-level language guide to determine which operations are allowed by the language.

Read Next: Positions the file to and gets the next record that is not deleted in the arrival sequence access path. Deleted records between the current position in the file and the next active record are skipped. (The READ statement in RPG/400 and the READ NEXT statement in COBOL/400 are examples of this operation.)

Read Previous: Positions the file to and gets the previous active record in the arrival sequence access path. Deleted records between the current file position and the previous active record are skipped. (The READP statement in RPG/400 and the READ PRIOR statement in COBOL/400 are examples of this operation.)

Read First: Positions the file to and gets the first active record in the arrival sequence access path.

Read Last: Positions the file to and gets the last active record in the arrival sequence access path.

Read Same: Gets the record that is identified by the current position in the file. The file position is not changed.

Read by Relative Record Number: Positions the file to and gets the record in the arrival sequence access path that is identified by the relative record number. The relative record number must identify an active record and must be less than or equal to the largest active relative record number in the member. This operation also reads the record in the arrival sequence access path identified by the current file position plus or minus some number of records. (The CHAIN statement in RPG/400 and the READ statement in COBOL/400 are examples of this operation.)

Reading Data Base Records Using a Keyed Sequence Access Path

The system performs the following read operations based on the statements you specify using your high-level language. These operations can be used with a keyed sequence access path to get data base records.

When a keyed sequence access path is used, a read operation cannot position to the storage occupied by a deleted record.

Note: Your high-level language may not allow all of the following operations. Refer to your high-level language guide to determine which operations are allowed by the language.

Read Next: Gets the next record on the keyed access path. If a record format name is specified, this operation gets the next record in the keyed access path that matches the record format. The current position in the file is used to locate the next record. (The READ statement in RPG/400 and the READ NEXT statement in COBOL/400 are examples of this operation.)

Read Previous: Gets the previous record on the keyed access path. If a record format name is specified, this operation gets the previous record on the keyed access path that matches the record format. The current position in the file is used to locate the previous record. (The READP statement in RPG/400 and the READ PRIOR statement in COBOL/400 are examples of this operation.)

Read First: Gets the first record on the keyed access path. If a record format name is specified, this operation gets the first record on the access path with the specified format name.

Read Last: Gets the last record on the keyed access path. If a record format name is specified, this operation gets the last record on the access path with the specified format name.

Read Same: Gets the record that is identified by the current file position. The position in the file is not changed.

Read by Key: Gets the record identified by the key value. Key operations of equal, equal or after, equal or before, read previous key equal, read next key equal, after, or before can be specified. If a format name is specified, the system searches for a record of the specified key value and record format name. If a format name is not specified, the entire keyed access path is searched for the specified key value. If the key definition for the file includes multiple key fields, a partial key can be specified (you can specify either the number of key fields or the key length to be used). This allows you to do generic key searches. If the program does not specify a number of key fields, the system assumes a default number of key fields. This default varies depending on if a record format name is passed by the program. If a record format

name is passed, the default number of key fields is the total number of key fields defined for that format. If a record format name is not passed, the default number of key fields is the maximum number of key fields that are common across all record formats in the access path. The program must supply enough key data to match the number of key fields assumed by the system. (The CHAIN statement in RPG/400 and the READ statement in COBOL/400 are examples of this operation.)

Read by Relative Record Number: For a keyed access path, the relative record number can be used. This is the relative record number in the arrival sequence, even though the member opened has a keyed access path. If the member contains multiple record formats, a record format name must be specified. In this case, you are requesting a record in the associated physical file member that matches the record format specified. If the member opened contains select/omit statements and the record identified by the relative record number is omitted from the keyed access path, an error message is sent to your program and the operation is not allowed. After the operation is completed, the file is positioned to the key value in the keyed access path that is contained in the physical record, which was identified by the relative record number. This operation also gets the record in the keyed sequence access path identified by the current file position plus or minus some number of records. (The CHAIN statement in RPG/400 and the READ statement in COBOL/400 are examples of this operation.)

Waiting for More Records When End of File Is Reached

End-of-file delay is a method of continuing to read sequentially from a data base file (logical or physical) after an end-of-file condition occurs. When an end-of-file condition occurs on a file being read sequentially (for example, next/previous record) and you have specified an end-of-file delay time (EOFDLY parameter on the Override with Data Base File [OVRDBF] command), the system waits for the time you specified. At the end of the delay time, another read is done to determine if any new records were added to the file. If records were added, normal record processing is done until an end-of-file condition occurs again. If records were not added to the file, the system waits again for the time specified.

For end-of-file delay times equal to or greater than 10 seconds, the job is eligible to be removed from main storage during the wait time. If you do not want the job eligible to be moved from main storage, specify PURGE(*NO) on the Create Class (CRTCLS) command for the CLASS the job is using.

To indicate which jobs have an end-of-file delay in progress, the status field of the Work with Active Jobs (WRKACTJOB) display shows an end-of-file wait or end-of-file activity level for jobs that are waiting for a record.

If a job uses end-of-file-delay and commitment control, it may hold its record locks for a longer period of time. This increases the chances that some other job may try to access those same records and be locked out. For that reason, be careful when using end-of-file-delay and commitment control in the same job.

If a file is shared, the Override with Data Base File (OVRDBF) command specifying an end-of-file delay must be requested before the *first* open of the file because overrides are ignored that are specified after the shared file is opened.

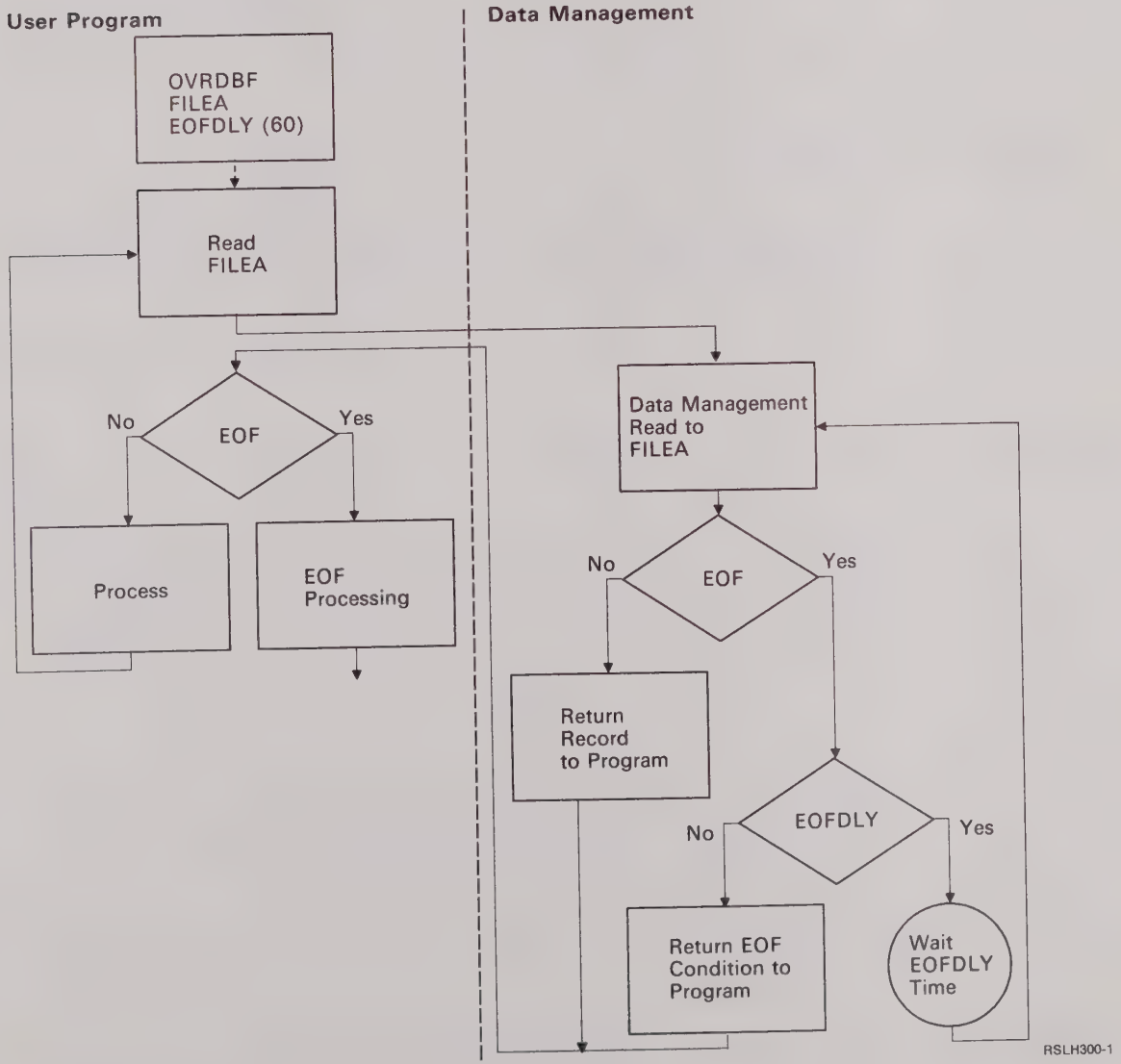
There are several ways to end a job that is waiting for more records because of an end-of-file-delay specified on the Override with Data Base File (OVRDBF) command:

- Write a record to the file with the end-of-file-delay that will be recognized by the application program as a last record. The application program can then specify

a force-end-of-data (FEOD) operation. An FEOD operation allows the program to complete normal end-of-file processing.

- Do a controlled end of a job by specifying OPTION(*CNTRLD) on the End Job (ENDJOB) command, with a DELAY parameter value time greater than the EOFDLY time. The DELAY parameter time specified must allow time for the EOFDLY time to run out, time to process any new records that have been put in the file, and any end-of-file processing required in your application. After new records are processed, the system signals end of file, and a normal end-of-file condition occurs.
- Specify OPTION(*IMMED) on the End Job (ENDJOB) command. No end-of-file processing is done.
- If the job is interactive, press the System Request key to end the last request.

The following is an example of end-of-file delay operation:



The actual processing of the EOFDLY parameter is more complex than shown because it is possible to force a true end-of-file if OPTION(*CNTRLD) on the End Job (ENDJOB) command is used with a long delay time.

The job does not become active whenever a new record is added to the file. The job becomes active after the specified end-of-file delay time ends. When the job becomes active, the system checks for any new records. If new records were added, the application program gets control and processes all new records, then waits again. Because of this, the job takes on the characteristic of a batch job when it is processing. For example, it normally processes a batch of requests. When the batch is completed, the job becomes inactive. If the delay is small, you may cause excessive system overhead because of the internal processing required to start the job and check for new records. Normally, only a small amount of overhead is used for a job waiting during end-of-file delay.

Note: When the job is inactive (waiting) it is in a long-wait status, which means it was released from an activity level. After the long-wait status is satisfied, the system reschedules the job in an activity level. (See the *Work Management Guide* for more information about activity levels.)

Releasing Locked Records

The system automatically releases a locked record when the record is updated, deleted, or when you read another record in the file. However, you may want to release a locked record without performing these operations. Some high-level languages support an operation to release a locked record. See your high-level language guide for more information about releasing record locks.

Note: The rules for locking are different if your job is running under commitment control. For more information about commitment control, see the *Backup and Recovery Guide*.

Updating Data Base Records

The update operation allows you to change an existing data base record in a logical or physical file. (The UPDAT statement in RPG/400 and the REWRITE statement in COBOL/400 are examples of this type operation.) Before you update a data base record, the record must first be read and locked. The lock is obtained by specifying the update option on any of the read operations listed under the "Reading Data Base Records Using an Arrival Sequence Access Path" on page 10-2 and "Reading Data Base Records Using a Keyed Sequence Access Path" on page 10-3.

If you issue several read operations with the update option specified, each read operation releases the lock on the previous record before attempting to locate and lock the new record. When you do the update operation, the system assumes that you are updating the currently locked record. Therefore, you do not have to identify the record to be updated on the update operation. After the update operation is done, the system releases the lock.

Note: The rules for locking are different if your job is running under commitment control. See the *Backup and Recovery Guide* for more details about commitment control.

If the update operation changes a key field in an access path for which immediate maintenance is specified, the access path is updated if the high-level language allows it. (Some high-level languages do not allow changes to the key field in an update operation. See your high-level language guide for more details.)

If you request a read operation on a record that is already locked for update, you must wait until the record is released or the time specified by the WAITRCD parameter that was specified on the create file or override commands has been exceeded. If the WAITRCD time is exceeded without the lock being released, an exception is returned to your program and a message is sent to your job stating the file, member, relative record number, and the job which has the lock.

Adding Data Base Records

The write operation is used to add a new record to a physical data base file member. (The WRITE statement in RPG/400 and the WRITE statement in COBOL/400 are examples of this operation.) New records can be added to a physical file member or to a logical file member that is based on the physical file member. When using a multiple format logical file, a record format name must be supplied to tell the system which physical file member to add the record to.

The new record is normally added at the end of the physical file member. The next available relative record number (including deleted records) is assigned to the new record. Some high-level languages allow you to write a new record over a deleted record position (for example, the WRITE statement in COBOL/400 when the file organization is defined as RELATIVE). For more information about writing records over deleted record positions, see your high-level language guide.

If you are adding new records to a file member that has a keyed access path, the new record appears in the keyed access path immediately at the location defined by the record key. If you are adding records to a logical member that contains select/omit values, the omit values may prevent the new record from appearing in the member's access path.

The SIZE parameter on the Create Physical File (CRTPF) and Create Source Physical File (CRTSRCPF) commands determines how many records can be added to a physical file member. For more information on this parameter, see the create data base file commands (CRTPF and CRTSRCPF) in the *CL Reference*.

Identifying Which Record Format to Add in a File with Multiple Formats

If your application uses a file name instead of a record format name for records to be added to the data base, and if the file used is a logical file with more than one record format, you need to write a format selector program to determine where a record should be placed in the data base. A format selector can be a CL program or a high-level language program.

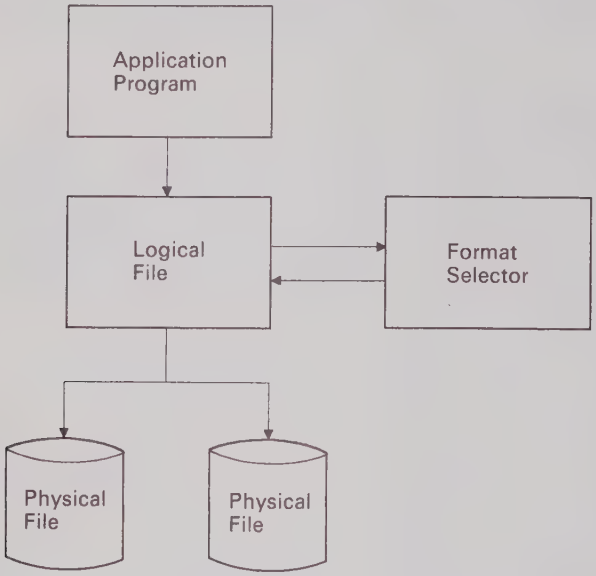
A format selector program must be used if all of the following are true:

- The logical file is not a join and not a view logical file.
- The logical file is based on multiple physical files.
- The program uses a file name instead of a record format name on the add operation.

If you do not write a format selector program for this situation, your program ends with an error when it tries to add a record to the data base.

Note: A format selector program cannot be used to select a member if a file has multiple members; it can only select a record format.

When an application program wants to add a record to the data base file, the system calls the format selector program. The format selector program examines the record and specifies the record format to be used. The system then adds the record to the data base file using the specified record format name.



The following example shows the programming statements for a format selector program written in RPG/400:

```
*ENTRY  PLIST
        PARM                    RECORD 80 (length of longest
                                          record expected)
        PARM                    FORMAT 10
        MOVELRECORD             BYTE 1
BYTE    IFEQ 'A'
        MOVEL'HDR'              FORMAT
        ELSE
        MOVEL'DTL'              FORMAT
        END
        RETRN
```

The format selector receives the record in the first parameter; therefore, this field must be declared to be the length of the longest record expected by the format selector. The format selector can access any portion of the record to determine the record format name. In this example, the format selector checks the first character in the record for the character A. If the first character is A, the format selector moves the record format name HDR into the second parameter (FORMAT). If the character is not A, the format selector moves the record format name DTL into the second parameter.

The format selector uses the second parameter, which is a 10-character field, to pass the record format name to the system. When the system knows the name of the record format, it adds the record to the data base.

You do not need a format selector if:

- You are doing updates only. For updates, your program already retrieved the record, and the system knows which physical file the record came from.
- Your application program specifies the record format name instead of a file name for an add or delete operation.
- All the records used by your application program are contained in one physical file.

To create the format selector, you use the create program command for the language in which you wrote the program. You cannot specify USRPRF(*OWNER) on the create command. The format selector must run under the user's user profile not the owner's user profile.

In addition, for security and integrity and because performance would be severely affected, you must not have any calls or input/output operations within the format selector.

The name of the format selector is specified on the FMTSLR parameter of the Create Logical File (CRTLF), Change Logical File (CHGLF), or Override with Data Base File (OVRDBF) command. The format selector program does not have to exist when the file is created; but it must exist when the application program is run.

Using the Force-End-Of-Data Operation

The force-end-of-data (FEOD) operation allows you to force all changes to a file made by your program to auxiliary storage. Normally, the system determines when to force changes to auxiliary storage. However, you can use the FEOD operation to ensure that all changes are forced to auxiliary storage. Some high-level languages do not support the FEOD operation. See your high-level language guide for more information about the FEOD operation.

Deleting Data Base Records

The delete operation allows you to delete an existing data base record. (The DELET statement in RPG/400 and the DELETE statement in COBOL/400 are examples of this operation.) To delete a data base record, the record must first be read and locked. The record is locked by specifying the update option on any of the read operations listed under "Reading Data Base Records" on page 10-2. The rules for locking records for deletion and identifying which record to delete are the same as for update operations.

Note: Some high-level languages do *not* require that you read the record first. These languages allow you to simply specify which record you want deleted on the delete statement. For example, RPG/400 allows you to delete a record without first reading it.

When a data base record is deleted, the physical record is marked as deleted. This is true even if the delete operation is done through a logical file. A deleted record *cannot* be read. The record is removed from all keyed access paths that contain the record. The relative record number of the deleted record remains the same. All other relative record numbers within the physical file member do not change.

The space used by the deleted record remains in the file, but it is not reused until:

- The Reorganize Physical File Member (RGZPFM) command is run to compress and free these spaces in the file member. See "Reorganizing Data in Physical File Members" on page 13-3 for more information about this command.
- Your program writes a record to the file by relative record number and the relative record number used is the same as that of the deleted record.

The system does not allow you to retrieve the data for a deleted record. You can, however, write a new record to the position (relative record number) associated with a deleted record. The write operation replaces the deleted record with a new record. See your high-level language guide for more details about how to write a record to a specific position (relative record number) in the file.

In order to write a record to the relative record number of a deleted record, that relative record number must exist in the physical file member. You can delete a record in the file by using the delete operation in your high-level language. You can also delete records in your file by using the the Initialize Physical File Member (INZPFM) command. The INZPFM command can initialize the entire physical file member to deleted records. For more information about the INZPFM command, see "Initializing Data in a Physical File Member" on page 13-2.

Chapter 11. Closing a Data Base File

When your program completes processing a data base file member, it should close the file. Closing a data base file disconnects your program from the file. The close operation releases all record locks and releases all file member locks, forces all changes made through the open data path (ODP) to auxiliary storage, then destroys the ODP. (When a shared file is closed but the ODP remains open, the functions differ. For more information about shared files, see "Sharing Data Base Files in the Same Job" on page 8-7.)

There are several ways you can close a data base file in a program, including:

- High-level language close statements
- Close File (CLOF) command
- Reclaim Resources (RCLRSC) command

Most high-level languages allow you to specify that you want to close your data base files. For more information about how to close a data base file in a high-level language program, see your high-level language guide.

You can use the Close File CLOF command to close data base files that were opened using either the Open Data Base File (OPNDBF) or Open Query File (OPNQRYF) commands.

You can also close data base files by running the Reclaim Resources (RCLRSC) command. The RCLRSC command releases all locks (except, under commitment control, locks on records that were changed but not yet committed), forces all changes to auxiliary storage, then destroys the open data path for that file. You can use the RCLRSC command to allow a calling program to close a called program's files. (For example, if the called program returns to the calling program without closing its files, the calling program can then close the called program's files.) However, the normal way of closing files in a program is with the high-level language close operation or through the Close File (CLOF) command. For more information about the RCLRSC command, see the *CL Reference*.

If a job ends normally (for example, a user signs off) and all the files associated with that job were not closed, the system automatically closes all the remaining open files associated with that job, forces all changes to auxiliary storage, and releases all record locks for those files. If a job ends abnormally, the system also closes all files associated with that job, releases all record locks for those files, and forces all changes to auxiliary storage.

Chapter 12. Handling Data Base File Errors in a Program

Error conditions that are detected during processing of a data base file cause messages to be sent to the program message queue for the program processing the file or cause an inquiry message to be sent to the system operator message queue. In addition, file errors and diagnostic information generally appear to your program as return codes and status information in the file feedback area. (For example, COBOL/400 sets a return code in the file status field, if it is defined in the program.) For more information about handling file errors in your program, see your high-level language guide.

If your programming language allows you to monitor for error messages, you can choose which ones you wish to monitor for. The following messages are some of the error messages you can monitor (see your high-level language guide and the *CL Reference* for a complete list of errors and messages you can monitor):

Message Identifier	Description
CPF5001	End of file reached
CPF5006	Record not found
CPF5007	Record deleted
CPF5018	Maximum file size reached
CPF5025	Read attempted past *START or *END
CPF5026	Duplicate key
CPF5027	Record in use by another job
CPF5028	Record key changed
CPF5029	Data mapping error
CPF5030	Partial damage on member
CPF5031	Maximum number of record locks exceeded
CPF5032	Record already allocated to job
CPF5033	Select/omit error
CPF5034	Duplicate key in another member's access path
CPF5040	Omitted record not retrieved
CPF5072	Join value in member changed
CPF5079	Commitment control resource limit exceeded
CPF5084	Duplicate key for uncommitted key
CPF5085	Duplicate key for uncommitted key in another access path
CPF5090	Invalid unique access path prevented add or update
CPF5097	Key mapping error

Note: To display the full description of these messages, use the Display Message Description (DSPMSGD) command.

If you do not monitor for any of these messages, the system handles the error. The system also sets the appropriate error return code in the program. Depending on the error, the system may end the job or send a message to the operator requesting further action.

If a message is sent to your program while processing a data base file member, the position in the file is not lost. It remains at the record it was positioned to before the message was sent, except:

- After an end-of-file condition is reached and a message is sent to the program, the file is positioned at *START or *END.
- After a conversion mapping message on a read operation, the file is positioned to the record containing the data that caused the message.

Part 4. Managing Data Base Files

Chapter 13. Managing Data Base Members

Before you can perform any input or output operations on a file, the file must have at least one member. As a general rule, data base files have only one member, the one created when the file is created. The name of this member is the same as the file name, unless you give it a different name. Because most operations on data base files assume that the member being used is the first member in the file, and because most files only have one member, you do not normally have to be concerned with, or specify, member names.

If there is more than one member in a file, each member serves as a subset of the data in the file. This allows you to classify data easier. For example, you define an accounts receivable file. You decide that you want to keep data for a year in that file, but you frequently want to process data just one month at a time. For example, you create a physical file with 12 members, one named for each month. Then, you process each month's data separately (by individual member). You can also process several or all members together.

Member Operations Common to All Data Base Files

The system supplies a way for you to:

- Add new members to an existing file.
- Change some attributes for an existing member (for example, the text describing the member) without having to recreate the member.
- Rename a member.
- Remove the member from the file.

The following section discusses these operations.

Adding Members to Files

There are a number of ways to add members to files:

- Automatically. When a file is created using the Create Physical File (CRTPF) and Create Logical File (CRTLF) commands, the default is to automatically add a member (with the same name as the file) to the newly created file. (The default for the Create Source Physical File [CRTSRCPF] command is *not* to add a member to the newly created file.) You can specify a different member name using the MBR parameter on the create data base file commands. If you do not want a member added when the file is created, specify *NONE on the MBR parameter.
- Specifically. After the file is created, you can add a member using the Add Physical File Member (ADDPFM) or Add Logical File Member (ADDLFM) commands.
- Copy File command. Another way to add members to a data base file is by using the Copy File (CPYF) command. If the member you are copying does not exist in the file being copied to, then the member is added to the file by the CPYF command.

Changing Member Attributes

You can use the Change Physical File Member (CHGPFM) or Change Logical File Member (CHGLFM) command to change certain attributes of a physical or a logical file member. For a physical file member, you can change the following parameters: EXPDATE (the member's expiration date), SHARE (whether the member can be shared within a job), and TEXT (the text description of the member). For a logical file member you can change the SHARE and TEXT parameters.

Note: You can use the Change Physical File (CHGPF) and Change Logical File (CHGLF) commands to change many other file attributes. For example, to change the maximum size allowed for each member in the file, you would use the SIZE parameter on the CHGPF command.

Renaming Members

The Rename Member (RNMM) command changes the name of an existing member in a physical or logical file. The file name is not changed.

Removing Members from Files

The Remove Member (RMVM) command is used to remove the member and its contents. Both the member data and the member, itself, are removed. After the member is removed, it can no longer be used by the system. This is different from just clearing or deleting the data from the member. If the member still exists, programs can continue to use (for example, add data to) the member.

Physical File Member Operations

The following section describes member operations that are unique to physical file members. Those operations include initializing data, clearing data, reorganizing data, and displaying data in a physical file member.

Initializing Data in a Physical File Member

To use relative record processing in a program, the data base file must contain a number of record positions equal to the highest relative record number used in the program. Programs using relative-record-number processing sometimes require that these records be initialized.

You can use the Initialize Physical File Member (INZPFM) command to initialize members with one of two types of records:

- Default records
- Deleted records

You specify which type of record you want using the RECORDS parameter on the Initialize Physical File Member (INZPFM) command.

If you initialize records using default records, the fields in each new record are initialized to the default field values defined when the file was created (see the DFT keyword in the *DDS Reference*). If no default field value was defined, then numeric fields are filled with zeros and character fields are filled with blanks.

Note: You cannot initialize a default record if the UNIQUE keyword is specified in DDS for the physical file member or any associated logical file members. Otherwise, you would create a series of duplicate key records.

If the records are initialized to the default records, you can read a record by relative record number and change the data.

If the records were initialized to deleted records, you can change the data by adding a record using a relative record number of one of the deleted records. (You cannot add a record using a relative record number that was not deleted.)

Deleted records cannot be read; they only hold a place in the member. A deleted record can be changed by writing a new record over the deleted record. Refer to "Deleting Data Base Records" on page 10-9 for more information about processing deleted records.

Clearing Data from Physical File Members

The Clear Physical File Member (CLRPFM) command is used to remove the data from a physical file member. After the clear operation is complete, the member description remains, but the data is gone.

Reorganizing Data in Physical File Members

You can use the Reorganize Physical File Member (RGZPFM) command to:

- Remove deleted records to make the space occupied by them available for more records.
- Reorganize the records of a file in the order in which you normally access them sequentially, thereby minimizing the time required to retrieve records. This is done using the KEYFILE parameter. This may be advantageous for files that are primarily accessed in an order other than arrival sequence. A member can be reorganized using either of the following:
  - Key fields of the physical file
  - Key fields of a logical file based on the physical file
- Reorganize a source file member, insert new source sequence numbers, and reset the source date fields (using the SRCOPT and SRCSEQ parameters on the Reorganize Physical File Member command).

For example, the following Reorganize Physical File Member (RGZPFM) command reorganizes the first member of a physical file using an access path from a logical file:

```
RGZPFM  FILE(DSTPRODLB/ORDHDRP)
        KEYFILE(DSTPRODLB/ORDFILL  ORDFILL)
```

The physical file ORDHDRP has an arrival sequence access path. It was reorganized using the access path in the logical file ORDFILL. Assume the key field is the *Order* field. The following illustrates how the records were arranged.

The following is an example of the original ORDHDRP file. Note that record 3 was deleted before the RGZPFM command was run:

Relative Record Number	Cust	Order	Ordate. . .
1	41394	41882	072480. . .
2	28674	32133	060280. . .
3	deleted	record	
4	56325	38694	062780. . .

The following example shows the ORDHDRP file reorganized using the *Order* field as the key field in ascending sequence:

Relative Record Number	Cust	Order	Ordate. . .
1	28674	32133	060280. . .
2	56325	38694	062780. . .
3	41394	41882	072480. . .

Notes:

1. If a file with an arrival sequence access path is reorganized using a keyed access path, the arrival sequence access path is changed. That is, the records in the file are physically placed in the order of the keyed sequence access path used. By reorganizing the data into a physical sequence that closely matches the keyed access path you are using, you can improve the performance of processing the data sequentially.
2. Reorganizing a file compresses deleted records, which changes subsequent relative record numbers.
3. Because access paths with either the FIFO or LIFO DDS keyword specified depend on the physical sequence of records in the physical file, the sequence of the records with duplicate key fields may change after reorganizing a physical file using a keyed sequence access path.

If one of the following conditions occur and the Reorganize Physical File Member (RGZPFM) command is running, the records may not be reorganized:

- The system ends abnormally.
- The job containing the RGZPFM command is ended with an *IMMED option.
- The subsystem in which the RGZPFM command is running ends with an *IMMED option.
- The system stops with an *IMMED option.

The status of the member being reorganized depends on how much the system was able to do before the reorganization was ended and what you specified in the SRCOPT parameter. If the SRCOPT parameter was not specified, the member is either completely reorganized or not reorganized at all. You should display the contents of the file, using the Display Physical File Member (DSPPFM) command, to determine if it was reorganized. If the member was not reorganized, issue the Reorganize Physical File Member (RGZPFM) command again.

If the SRCOPT parameter was specified, any of the following could have happened to the member:

- It was completely reorganized. A completion message is sent to your job log indicating the reorganize operation was completely successful.
- It was not reorganized at all. A message is sent to your job log indicating that the reorganize operation was not successful. If this occurs, issue the Reorganize Physical File Member (RGZPFM) command again.
- It was reorganized, but only some of the sequence numbers were changed. A completion message is sent to your job log indicating that the member was reorganized, but all the sequence numbers were not changed. If this occurs, issue the RGZPFM command again with KEYFILE(*NONE) specified.

Displaying Records in a Physical File Member

The Display Physical File Member (DSPPFM) command can be used to display the data in the physical data base file members by arrival sequence. The command can be used for:

- Problem analysis
- Debugging
- Record inquiry

You can display source files or data files, regardless if they are keyed or arrival sequence. Records are displayed in arrival sequence, even if the file is a keyed file. You can page through the file, locate a particular record by record number, or shift the display to the right or left to see other parts of the records. You can also press a function key to show either character data or hexadecimal data on the display.

If you have Query installed, you can use the Start Query (STRQRY) command to select and display records, too. For more information about Query, refer to the *Query User's Guide*.

If you have the SQL/400 installed, you can use the Start SQL/400 (STRSQL) command to interactively select and display records. For more information about SQL/400, see the *SQL/400 Programmer's Guide* and the *SQL/400 Reference*.

Chapter 14. Changing Data Base File Descriptions and Attributes

This chapter describes the things to consider when planning to change the description or attributes of a data base file.

Effect of Changing Fields in a File Description

When a program that uses externally described data is compiled, the compiler copies the file descriptions of the files into the compiled program. When you run the program, the system can verify that the record formats the program was compiled with are the same as the record formats currently defined for the file. The default is to do level checking.

The system assigns a unique level identifier for each record format when the file it is associated with is created. The system uses the information in the record format description to determine the level identifier. This information includes the total length of the record format, the record format name, the number and order of fields defined, the data type, the size of the fields, the field names, and the number of decimal positions in the field. Changes to this information in a record format cause the level identifier to change.

The following DDS keywords have no effect on the level identifier and, therefore, can be changed without recompiling the program that uses the file:

- TEXT keyword
- COLHDG keyword
- CHECK keyword
- EDTCDE keyword
- EDTWRD keyword
- REF keyword
- REFFLD keyword
- CMP, RANGE, and VALUES keywords
- Key fields
- Access path keywords
- Select/omit fields

Keep in mind that even though changing key fields or select/omit fields will not cause a level check, the change may cause unexpected results in programs using the new access path. For example, changing the key field from the customer number to the customer name changes the order in which the records are retrieved, and may cause unexpected problems in the programs processing the file.

If level checking is specified (or defaulted to), the level identifier of the file to be used is compared to the level identifier of the file in your program when the file is opened. If the identifiers differ, a message is sent to the program to identify the condition that the format was changed and the changes may affect your program. You can simply compile your program again so that the changes are included.

An alternative is to display the file description to determine if the changes affect your program. You can use the Display File Field Description (DSPFFD) command

to display the description or, if you have SEU, you can display the source file containing the DDS for the file.

The format level identifier defined in the file can be displayed by the Display File Description (DSPFD) command. When you are displaying the level identifier, remember that the record format identifier is compared, rather than the file identifier.

Not every change in a file necessarily affects your program. For example, if you add a field to the end of a file and your program does not use the new field, you do not have to recompile your program. If the changes do not affect your program, you can use the Change Physical File (CHGPF) or the Change Logical File (CHGLF) commands with LVLCHK(*NO) specified to turn off level checking for the file, or you can enter an Override with Data Base File (OVRDBF) command with LVLCHK(*NO) specified so that you can run your program without level checking.

Keep in mind that level checking is the preferred method of operating. The use of LVLCHK(*YES) is a good data base integrity practice. The results produced by LVLCHK(*NO) cannot always be predicted.

Changing a Physical File Description and Attributes

Sometimes, when you make a change to a physical file description and then recreate the file, the level identifier can change. For example, the level identifier will change if you add a field to the file description, or change the length of an existing field. If the level identifier changes, you can compile the programs again that use the physical file. After the programs are recompiled, they will use the new level check identifier.

You can avoid compiling again by creating a logical file that presents data to your programs in the original record format of the physical file. Using this approach, the logical file has the same level check identifier as the physical file before the change.

For example, you decide to add a field to a physical file record format. You can avoid compiling your program again by doing the following:

1. Change the DDS and create a new physical file (FILEB in LIBA) to include the new field:

   ```
   CRTPF FILE(LIBA/FILEB) MBR(*NONE)...
   ```

   FILEB does not have a member. (The old file FILEA is in library LIBA and has one member MBRA.)
2. Copy the member of the old physical file to the new physical file:

   ```
   CPYF    FROMFILE(LIBA/FILEA) TOFILE(LIBA/FILEB)
           FROMMBR(*ALL) TOMBR(*FROMMBR)
           MBROPT(*ADD) FMTOPT(*MAP)
   ```

   The member in the new physical file is automatically named the same as the member in the old physical file because FROMMBR(*ALL) and TOMBR(*FROMMBR) are specified. The FMTOPT parameter specifies to copy (*MAP) the data in the fields by field name.
3. Describe a new logical file (FILEC) that looks like the original physical file (the logical file record format does *not* include the new physical file field). Specify FILEB for the PFILE keyword. (When a level check is done, the level identifier in the logical file and the level identifier in the program match because FILEA and FILEC have the same format.)

4. Create the new logical file:

   ```
   CRTLF FILE(LIBA/FILEC)...
   ```

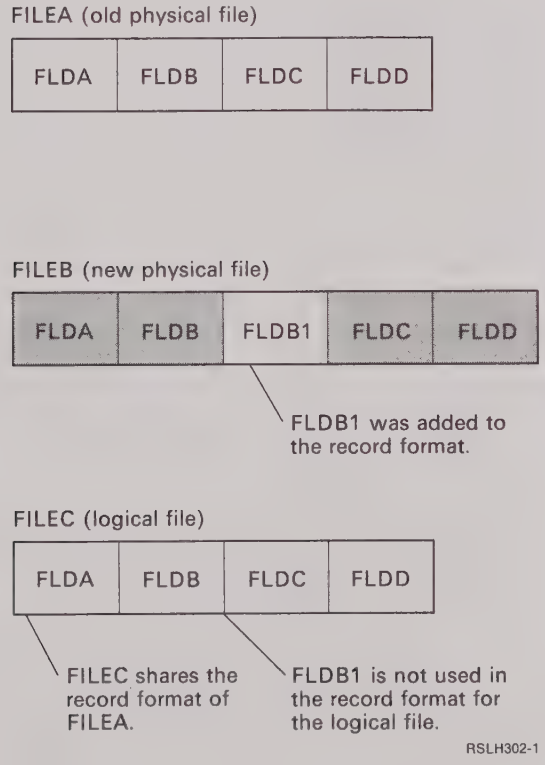
5. You can now do one of the following:
   a. Use an Override with Data Base File (OVRDBF) command in the appropriate jobs to override the old physical file referred to in the program with the logical file (the OVRDBF command parameters are described in more detail in Chapter 8).

      ```
      OVRDBF FILE(FILEA) TOFILE(LIBA/FILEC)
      ```

   b. Delete the old physical file and rename the logical file to the name of the old physical file so the file name in the program does not have to be overridden.

      ```
      DLTF FILE(LIBA/FILEA)
      RNMOBJ   OBJ(LIBA/FILEC) OBJTYPE(*FILE)
               NEWOBJ(FILEA)
      ```

The following illustrates the relationship of the record formats used in the three files:



When you make changes to a physical file that cause you to create the file again, all logical files referring to it must first be deleted before you can delete and create the new physical file. After the physical file is recreated, you can recreate or restore the logical files referring to it. The following examples show two ways to do this:

Example 1: Create a new physical file with the same name in a different library

1. Create a new physical file with a different record format in a library different from the library the old physical file is in. The name of new file should be the

same as the name of the old file. (The old physical file FILEPFC is in library LIBB and has two members, MBRC1 and MBRC2.)

```
CRTPF FILE(NEWLIB/FILEPFC) MAXMBRS(2)...
```

2. Copy the members of the old physical file to the new physical file. The members in the new physical file are automatically named the same as the members in the old physical file because TOMBR(*FROMMBR) and FROMMBR(*ALL) are specified.

```
CPYF        FROMFILE(LIBB/FILEPFC) TOFILE(NEWLIB/FILEPFC)
            FROMMBR(*ALL) TOMBR(*FROMMBR)
            FMTOPT(*MAP *DROP) MBROPT(*ADD)
```

3. Describe and create a new logical file in a library different from the library the old logical file is in. The name of the new logical file should be the same as the old logical file name. (You can use the FORMAT keyword to use the same record formats as are in the current logical file, if no changes need be made to the record formats.) You can also use the Create Duplicate Object (CRTDUPOBJ) command to create the new logical file. The old logical file FILELFC is in library LIBB.

```
CRTLF FILE(NEWLIB/FILELFC)
```

4. Delete the old logical and physical files.

```
DLTF FILE(LIBB/FILELFC)
DLTF FILE(LIBB/FILEPFC)
```

5. Move the newly created files to the original library by using the following commands:

```
MOVOBJ OBJ(NEWLIB/FILELFC) OBJTYPE(*FILE) TOLIB(LIBB)
MOVOBJ OBJ(NEWLIB/FILEPFC) OBJTYPE(*FILE) TOLIB(LIBB)
```

Example 2: Creating new versions of files in the same libraries

1. Create a new physical file with a different record format in the same library the old physical file is in. The names of the files should be different. (The old physical file FILEPFA is in library LIBA and has two members MBRA1 and MBRA2.)

```
CRTPF FILE(LIBA/FILEPFB) MAXMBRS(2)...
```

2. Copy the members of the old physical file to the new physical file.

```
CPYF        FROMFILE(LIBA/FILEPFA) TOFILE(LIBA/FILEPFB)
            FROMMBR(*ALL) TOMBR(*FROMMBR)
            FMTOPT(*MAP *DROP) MBROPT(*REPLACE)
```

3. Create a new logical file in the same library as the old logical file is in. The names of the old and new files should be different. (You can use the FORMAT keyword to use the same record formats as are in the current logical file if no changes need be made to the record formats.) The PFILE keyword must refer to the new physical file created in step 1. The old logical file FILELFA is in library LIBA.

```
CRTLF FILE(LIBA/FILELFB)
```

4. Delete the old logical and physical files.

```
DLTF FILE(LIBA/FILELFA)
DLTF FILE(LIBA/FILEPFA)
```

5. Rename the new logical file to the name of the old logical file. (If you also decide to rename the physical file, be sure to change the DDS for logical file so that the PFILE keyword refers to the new physical file name.)

   ```
   RNMOBJ(LIBA/FILELFB) OBJTYPE(*FILE) NEWOBJ(FILELFA)
   ```

6. If the logical file member should be renamed, and assuming the default was used on the Create Logical File (CRTLF) command, issue the following command:

   ```
   RNMM FILE(LIBA/FILELFA) MBR(FILELFB) NEWMBR(FILELFA)
   ```

You can use the Change Physical File (CHGPF) command to change some of the attributes of a physical file and its members. For information on these parameters, see the Change Physical File (CHGPF) command in the *CL Reference*.

Changing a Logical File Description and Attributes

As a general rule, when you make changes to a logical file that will cause a change to the level identifier (for example, adding a new field, deleting a field, or changing the length of a field), it is *strongly* recommended that you recompile the programs that use the logical file. Sometimes you can make changes to a file that change the level identifier and which do not require you to recompile your program (for example, adding a field that will not be used by your program to the end of the file). However, in those situations you will be forced to turn off level checking to run your program using the changed file. That is not the preferred method of operating. It increases the chances of incorrect data in the future.

To avoid recompiling, you can keep the current logical file (unchanged) and create a new logical file with the added field. Your program refers to the old file, which still exists.

You can use the Change Logical File (CHGLF) command to change most of the attributes of a logical file and its members that were specified on the Create Logical File (CRTLF) command. For information on these parameters, see the Change Logical File (CHGLF) command in the *CL Reference*.

Chapter 15. Using Data Base Attribute and Cross-Reference Information

The AS/400 integrated data base provides file attribute and cross-reference information. Some of the cross-reference information includes:

- The files used in a program
- The files that depend on other files for data or access paths
- File attributes
- The fields defined for a file

Each of the commands described in the following sections can present information on a display, a printout, or write the cross-reference information to a data base file that, in turn, can be used by a program or utility (for example, Query) for analysis.

For more information about writing the output to a data base file, see "Writing the Output from a Command Directly to a Data Base File" on page 15-4.

Displaying Information about Data Base Files

Displaying Attributes for a File

You can use the Display File Description (DSPFD) command to display the file attributes for data base files and device files. The information can be displayed, printed, or written to a data base file. The information supplied by this command includes (parameter values given in parentheses):

- Basic attributes (*BASATR)
- File attributes (*ATR)
- Access path specifications (*ACCPTH, logical and physical files only)
- Select/omit specifications (*SELECT, logical files only)
- Join file specifications (*JOIN, logical files only)
- Alternative collating sequence specifications (*SEQ, physical and logical files only)
- Record format specifications (*RCDFMT)
- Member attributes (*MBR, physical and logical files only)
- Spooling attributes (*SPOOL, printer and diskette files only)
- Member lists (*MBRLIST, physical and logical files only)

Displaying the Descriptions of the Fields in a File

You can use the Display File Field Description (DSPFFD) command to display field information for both data base and device files. The information can be displayed, printed, or written to a data base file.

Displaying the Relationships between Files on the System

You can use the Display Data Base Relations (DSPDBR) command to display the following information about the organization of your data base:

- A list of data base files (physical and logical) that use a specific record format.
- A list of data base files (physical and logical) that depend on the specified file for data sharing
- A list of members (physical and logical) that depend on the specified member for sharing data or sharing an access path.

This information can be displayed, printed, or written to a data base file.

For example, to display a list of all data base files associated with physical file ORDHDRP, with the record format ORDHDR, type the following DSPDBR command:

```
DSPDBR FILE(DSTPRODLB/ORDHDRP) RCDFMT(ORDHDR)
```

Note: See the DSPDBR command description in the *CL Reference* for details of this display.

This display presents header information when a record format name is specified on the RCDFMT parameter, and presents information about which files are using the specified record format.

If a member name is specified on the MBR parameter of the DSPDBR command, the dependent members are shown.

If the Display Data Base Relations (DSPDBR) command is specified with the default MBR(*NONE) parameter value, the dependent data files are shown. To display the shared access paths, you must specify a member name.

The Display Data Base Relations (DSPDBR) command output identifies the type of sharing involved using the following codes:

Code	Description
D	Data sharing. The file or member is dependent on the data in a member of another file.
I	Access path sharing. The file member is sharing an access path.
O	Access path owner. If an access path is shared, one of the file members is considered the owner. The owner of the access path is charged with the storage used for the access path. If the member displayed is designated the owner, one or more file members are designated with an I for access path sharing.

Note: If the results of the command are written to a data base file, then the code (type of sharing) is placed in the *Whtype* field in the records of the output file.

Displaying the Files Used by Programs

You can use the Display Program Reference (DSPPGMREF) command to determine which files, data areas, and other programs are used by a program. This information is available for compiled programs only.

The information can be displayed, printed, or written to a data base file.

When a program is created, the information about certain objects used in the program is stored. This information is then available for use with the Display Program References (DSPPGMREF) command.

The following chart shows the objects for which the high-level languages and utilities save information:

Language or Utility	Files	Programs	Data Areas	See Notes
BASIC	Yes	Yes	Yes	1
CL	Yes	Yes	Yes	2
COBOL/400	Yes	Yes	No	3
DFU	Yes	N/A	N/A	
Pascal	No	No	N/A	
PL/I	Yes	Yes	N/A	3
RPG/400	Yes	No	Yes	4
SQL/400	Yes	N/A	N/A	

Notes:

1. Externally described file references, programs, and data areas are stored.
2. All system commands that refer to files, programs, or data areas specify in the command definition that the information should be stored when the command is compiled in a CL program. If a variable is used, the name of the variable is used as the object name (for example, &FILE). If an expression is used, the name of the object is stored as *EXPR. User-defined commands can also store the information for files, programs, or data areas specified on the command. See the description of the FILE, PGM, and DTAARA parameters on the PARM or ELEM command statements in *CL Programmer's Guide*.
3. The program name is stored only when a literal is used for the program name (this is a static call, for example, CALL 'PGM1'), not when a COBOL/400 identifier is used for the program name (this is a dynamic call, for example, CALL PGM1).
4. The use of the local data area is not stored.

The stored file information contains an entry (a number) for the type of use. In the data base file output of the Display Program References (DSPPGMREF) command (built when using the OUTFILE parameter), this is specified as:

Code	Meaning
1	Input
2	Output
3	Input and Output
4	Update
8	Unspecified

Combinations of codes are also used. For example, a file coded as a 7 would be used for input, output, and update.

Displaying the System Cross-Reference Files

The system manages two data base files that contain basic file attribute information (QSYS/QADBXREF) and cross-reference information (QSYS/QADBFDEP) about all the data base files on the system. You can use these files to determine basic attribute and file dependencies. To display the fields contained in these files, use the Display File Field Description (DSPFFD) command.

Note: Normally, the authority to use these files is restricted to the security officer.

Writing the Output from a Command Directly to a Data Base File

You can store the output from many CL commands in an output physical file by specifying the OUTFILE parameter on the command.

You can use the output files in programs or utilities (for example, Query) for data analysis. For example, you can send the output of the Display Program References (DSPPGMREF) command to a physical file, then query that file to determine which programs use a specific file.

The physical files are created for you when you specify the OUTFILE parameter on the commands. Initially, the files are created with private authority; only the owner (the person who ran the command) can use it. However, the owner can authorize other users to these files as you would for any other data base file.

The system supplies model files that identify the record format for each command that can specify the OUTFILE parameter. If you specify a file name on the OUTFILE parameter for a file that does not already exist, the system will create the file using the same record format as the model files. If you specify a file name for an existing output file, the system checks to see if the record format is the same record format as the model file. If the record formats do not match, the system sends a message to the job and the command does not complete.

Note: You must use your own files for output files, rather than specifying the system-supplied model files on the OUTFILE parameter.

The following is a list of commands that allow output files, and the model files supplied for those commands:

Command	Description	Model File	Format Name
DSPAUTHLR	Display Authority Holder	QADSHLR	QSYDSHLR
DSPAUTL	Display Authority List	QAOBJAUT	QSYDSAUTH
DSPAUTLOBJ	Display Authority List Objects	QADALO	QSYDALO
DSPDBR	Display Data Base Relations	QADSPDBR	QWHDRDBR
DSPDIR[1]	Display Directory		
DSPDSTL[1]	Display Distribution List		
DSPFD[2]	Display File Description		
DSPFFD	Display File Field Description	QADSPFFD	QWHDRFFD
DSPFLR[1]	Display Folder		
DSPJRN[3]	Display Journal	QADSPJRN	QJORDJE
DSPOBJAUT	Display Object Authority	QAOBJAUT	QSYDSAUTH
DSPOBJD	Display Object Description	QADSPOBJ	QLIDOBJD
DSPPGMADP	Display Programs That Adopt	QADPGMMAD	QADPGMAD
DSPPGMREF	Display Program References	QADSPPGM	QWHDRPPR
DSPPTF	Display PTF	QADSPPTF	QSCPTF
PRTDOC[1]	Print Document		
PRTERRLOG	Print Error Log	QAPRTELG	QSCELOG
QRYDOCLIB[1]	Query Document Library		
QRYDST[1]	Query Distribution		
RCVDST[1]	Receive Distribution		
RTVDOC[1]	Retrieve Document		
RUNQRY[4]	Run Query		
STRCPYSCN	Start Screen Copy	QASCCPY	OSCCPY1
TRCJOB	Trace Job	QATRCJOB	QSCTJTRC
WRKNETF	Work with Network Files	QANFDNTF	QNFDNTF

1 See Appendix D for more details.

2 See "Output File for the Display File Description Command" on page 15-7.

3 Additional model files are supplied for specific types of journal entries. See "Output Files for the Display Journal Command" on page 15-8 for more information.

4 No model file is supplied.

Note: All system-supplied model files are located in the QSYS library.

You can display the fields contained in the record formats of the system-supplied model files by using the Display File Field Descriptions (DSPFFD) command.

Example of Using a Command Output File

The following example uses the Display Program References (DSPPGMREF) command to collect information for all compiled programs in all libraries, and place the output in a data base file named DBROUT:

```
DSPPGMREF  PGM(*ALL/*ALL)  OUTPUT(*OUTFILE)  OUTFILE(DSTPRODLB/DBROUT)
```

You can use Query to process the output file. Another way to process the output file is to create a logical file to select information from the file. The following is the DDS for such a logical file. Records are selected based on the file name.

```
     A*  Logical file DBROUTL   for query
     A
     A            R DBROUTL                  PFILE(DBROUT)
     A            S WHFNAM                   VALUES('ORDHDRL' 'ORDFILL')
     A
     A
```

RSLH301-0

Output File for the Display File Description Command

The Display File Description (DSPFD) command provides unique output files, depending on the parameters specified. The model files for the DSPFD command are as follows:

Type	File Attribute	File Name	Record Format Name
*ACCPTH		QAFDACCP	QWHFDACP
*BASATR		QAFDBASI	QWHFDBAS
*SEQ		QAFDCSEQ	QWHFDSEQ
*JOIN		QAFDJOIN	QWHFDJN
*MBR		QAFDMBR	QWHFDMBR
*MBRLIST		QAFDMBRL	QWHFDML
*RCDFMT		QAFDRFMT	QWHFDFMT
*SELECT		QAFDSELO	QWHFDSO
*SPOOL		QAFDSPOL	QWHFDSPL
*ATR	*BSCF	QAFDBSC	QWHFDBSC
*ATR	*CMNF	QAFDCMN	QWHFDCMN
*ATR	*DDMF	QAFDDDM	QWHFDDDM
*ATR	*DKTF	QAFDDKT	QWHFDDKT
*ATR	*DSPF	QAFDDSP	QWHFDDSP
*ATR	*ICFF	QAFDICF	QWHFDICF
*ATR	*LF	QAFDLGL	QWHFDLGL
*ATR	*PF	QAFDPHY	QWHFDPHY
*ATR	*PRTF	QAFDPRT	QWHFDPRT
*ATR	*SAVF	QAFDSAV	QWHFDSAV
*ATR	*TAPF	QAFDTAP	QWHFDTAP

Note: All system-supplied model files are in the QSYS library.

To collect access path information about all files in the LIBA library, you could specify:

```
DSPFD   FILE(LIBA/*ALL)  TYPE(*ACCPTH)  OUTPUT(*OUTFILE) +
          OUTFILE(LIBB/ABC)
```

The file ABC is created in library LIBB and is externally described with the same field descriptions as in the system-supplied file QSYS/QAFDACCP. The ABC file then contains a record for each key field in each file found in library LIBA that has an access path.

If the Display File Description (DSPFD) command is coded as:

```
DSPFD   FILE(LIBX/*ALL)  TYPE(*ATR)  OUTPUT(*OUTFILE) +
          FILEATR(*PF)  OUTFILE(LIBB/DEF)
```

the file DEF is created in library LIBB and is externally described with the same field descriptions as exist in QSYS/QAFDPHY. The DEF file then contains a record for each physical file found in library LIBX.

You can display the field names of each model file supplied by IBM by using the DSPFFD command. For example, to display the field description for the access path model file (*ACCPTH specified on the TYPE parameter), specify the following:

```
DSPFFD    QSYS/QAFDACCP
```

Output Files for the Display Journal Command

The following is a list of Display Journal (DSPJRN) command model output files supplied on the system.

Description	Model File	Record Format
Standard model file	QADSPJRN	QJORDJE
Job accounting	QAJBACG	QWTJAJBE
Print accounting	QAPTACG	QSPJAPTE
SNADS configuration	QAZDCFLG	QZDCFLOG
SNADS error log	QAZDERLG	QZDERLOG
SNADS logging	QAZDJRNL	QZDLGLOG
SNADS routing	QAZDRTLG	QZDRTLOG
SNADS alert routing	QAZDALLG	QZALLLOG
DSNX logging	QADXJRNL	QDXLGLOG
DSNX error logging	QADXERLG	QDXERLOG

Chapter 16. Planning Data Base Recovery

This chapter discusses several things to consider when you are planning for data base recovery. For detailed information about the AS/400 recovery functions, see the *Backup and Recovery Guide*.

Data Base Save and Restore Considerations

It is very important that you periodically save your files (and other objects). Saving your files (and other objects) is fundamental to protecting your data. This section describes the AS/400 save and restore functions as they apply to data base files. For more information about saving and restoring files and other objects on the system, see the *Backup and Recovery Guide*.

Save and Restore Commands

The AS/400 system has a full set of commands to save and restore your data base files. The save commands save data base files on tape or diskette. In addition, data base files can be saved in a disk-resident save file, then later written to tape or diskette. The save and restore commands used to save and restore files include:

- The Save Library (SAVLIB) command to save one or more libraries
- The Save Object (SAVOBJ) command to save one or more objects (including data base files and members)
- The Save Changed Object (SAVCHGOBJ) command to save any file that changed since either the last time the library was saved or from a specified date
- The Save Save File Data (SAVSAVFDTA) command to save the contents of a save file
- The Restore Library (RSTLIB) command to restore a library
- The Restore Object (RSTOBJ) command to restore one or objects (including data base files and members)

A detailed description of these commands can be found in the *Backup and Recovery Guide* and the *CL Reference*.

Saving and Restoring Data Base File Members

The save and restore commands can be used to save and restore your data base files. You also have an option on the SAVOBJ and RSTOBJ commands to save and restore one or more data base file members instead of the complete file. For more information on this option, see the description of the FILEMBR parameter on the SAVOBJ and RSTOBJ commands in the *CL Reference*.

Saving and Restoring Access Paths

Restoring an access path can be faster than rebuilding it. For example, assume a logical file is built over a physical file containing 500,000 records and you have determined (through the Display Object Description [DSPOBJD] command) that the size of the logical file is about 15 megabytes. (This assumes that the system can build approximately 10,000 index entries per minute.) In this example, assume it takes about 50 minutes to rebuild the access path for the logical file compared to about one minute to restore the same access path from a tape.

After restoring the access path, the file may need to be brought up-to-date by applying the latest journal changes (depending on whether journaling is active). Normally, the system can apply approximately 80,000 to 100,000 journal entries per hour. (This assumes that each of the physical files to which entries are being applied has only one access path built over it.) Even with this additional recovery time, you will usually find it is faster to restore access paths rather than to rebuild them.

There are other things to consider when saving and restoring access paths. You can find more information about saving and restoring access paths in the *Backup and Recovery Guide*.

Saving and Restoring Data Using a Disk-Resident Save File

Save files are disk resident files that can be the target of a save operation or the source of a restore operation. Save files allow unattended save operations. That is, an operator does not need to load tapes or diskettes when saving to a save file. However, it is still important to use the SAVSAVFDTA command to periodically save the save file data on tape or diskette. The tapes or diskettes should periodically be removed from the site. Storing a copy of your save tapes or diskettes off the site is important to help recover from a site disaster.

Save files can be used when you run the following commands:

- The Save Library (SAVLIB) command (except when saving multiple libraries)
- The Save Object (SAVOBJ) command
- The Save Changed Object (SAVCHGOBJ) command
- The Restore Library (RSTLIB) command
- The Restore Object (RSTOBJ) command

Data Base Data Recovery

The AS/400 system has integrated recovery functions and commands to help recover data in a data base file. One function discussed in this section is journal management. Journal management allows you to record all the data changes occurring to one or more data base files. You can then use the journal for recovery.

Journal Management

You should seriously consider using journal management. If a data base file is destroyed or becomes unusable and you are using journaling, you can reconstruct most of the activity for the file. Optionally, the journal allows you to remove changes made to the file.

Journaling can be started or ended very quickly. It requires no additional programming or changes to existing programs.

When a change is made to a file and you are using journaling, the system records the change in a journal receiver and writes the receiver to auxiliary storage before it is recorded in the file. Therefore, the journal receiver always has the latest data base information. Activity for a file is recorded regardless of the type of program, user, or job that made the change, or the logical file through which the change was made.

Journal receiver entries record activity for a specific record (record added, updated or deleted), or for the file as a whole (file opened, file member saved, and so on). Each entry includes additional bytes of control information identifying the source of the activity (including user, job, program, time, and date). For changes that affect a

single record, record images are included following the control information. The image of the record after a change is made is always included. Optionally, the record image before the change is made can also be included. You control whether to record both before and after record images or just after record images by specifying the IMAGES parameter on the Start Journaling Physical File (STRJRNPF) command.

All journaled data base files are automatically synchronized with the journal when the system is started (IPL time). If the system ended abnormally, some data base changes may be in the journal, but not yet reflected in the data base itself. If that is the case, the system automatically updates the data base from the journal to bring the data base files up to date.

Journaling can make saving data base files easier and faster. For example, instead of saving an entire file everyday, you can simply save the journal receiver that contains the changes to that file. You might still save the entire file on a weekly basis. This method can reduce the amount of time it takes to perform your daily save operations.

The Display Journal (DSPJRN) command, can be used to convert journal receiver entries to a data base file. Such a file can be used for activity reports, audit trails, security, and program debugging.

Because the journal supplies many useful functions, not the least of which is recovering data, journal management ought to be considered a key part of your recovery strategy.

The *Backup and Recovery Guide* contains additional information on journaling.

Writing Data to Auxiliary Storage

The force-write ratio (FRCRATIO) parameter on the create file and override data base file commands can be used to force data to be physically written to auxiliary storage. A force-write ratio of one causes every add, update, and delete request to be immediately written to auxiliary storage for the file in question. However, choosing this option can reduce system performance. Therefore, saving your files and journaling your files should be considered the primary methods for protecting data base files.

Transaction Recovery

The AS/400 system has an integrated transaction recovery function called commitment control. Commitment control is an extension of the journal function on the system. It provides you additional assistance in recovering data and restarting your programs. Commitment control can ensure that complex application transactions are logically synchronized even if the job or system ends abnormally.

Transactions can be classified as follows:

- Inquiries in which no file changes occur.
- Simple transactions in which one file is changed each time you press the Enter key.
- Complex transactions in which two or more files are changed each time you press the Enter key.

- Complex transactions in which one or more files are changed each time you press the Enter key, but these changes represent only part of a logical group of transactions.

If the system or job ends abnormally, journaling alone can ensure that, at most, only the very last record change is lost. However, if the system or job ends abnormally during a complex transaction (where more than one file may be changed), the files can reflect an incomplete logical transaction. For example, the job may have updated a record in file A, but before it had a chance to update a corresponding record in file B the job ended abnormally. In this case, the logical transaction consisted of two updates, but only one update completed before the job ended abnormally.

Recovering a complex application requires detailed application knowledge. Programs cannot simply be restarted. Adjustments may have to be made with an application program or data file utility to reverse the files to just before the last complex transaction began. This task becomes more complex if multiple users were accessing the files at the same time.

Commitment control helps solve these problems. Under commitment control, the records used during a complex transaction are locked from other users. This ensures that other users do not use the records until the transaction is complete. At the end of the transaction, the program issues the commit operation, freeing the records. However, should the system or job end abnormally before the commit operation is performed, all record changes for that job since the last time a commit operation occurred are rolled back. Any affected records that are still locked are then unlocked. In other words, data base changes are rolled back to a clean transaction boundary.

The rollback operation can also occur under your control. Assume that in an order entry application, the application program runs the commit operation at the end of each order. In the middle of an order, the operator can signal the program to do a rollback operation. All file changes will be rolled back to the beginning of the order.

The commit and roll back operations are available in several AS/400 programming languages including RPG/400, COBOL/400, PL/I, SQL/400, and the system control language.

An optional feature of commitment control is the use of a notify object. The notify object is a file, data area, or message queue. When a rollback operation occurs, information specified by the program is automatically sent to the notify object. This information can be used by an operator or application programs to start the application from the last successful transaction boundary.

Commitment control can also be used in a batch environment. Just as it provides assistance in interactive transaction recovery, commitment control can help in batch job recovery.

The *Backup and Recovery Guide* contains additional information about commitment control.

Access Path Recovery

The system ensures the integrity of an access path before you can use it. If the system determines that the access path is unusable, the system attempts to recover it. You can control when an access path will be recovered. See "Controlling When Access Paths Are Rebuilt" on page 16-7 for more information.

Access path recovery can be time consuming, especially if you have large access paths or many access paths to be rebuilt. The system can assist you in reducing this recovery time.

Rebuilding Access Paths

Rebuilding a data base access path takes approximately one minute for every 10,000 records.

Note: This estimate should be used until actual times for your system can be calculated.

The following factors affect this time estimate (listed in general order of significance):

- Storage pool size. The size of the storage pool used to rebuild the access path is a very important factor. You can improve the rebuild time by running the job in a larger storage pool.
- The system model. The speed of the processing unit is a key factor in the time needed to rebuild an access path.
- Key length. A large key length will slow rebuilding the access path because more key information must be constructed and stored in the access path.
- Select/omit values. Select/omit processing will slow the rebuilding of an access path because each record must be compared to see if it meets the select/omit values.
- Record length. A large record length will slow the rebuilding of an access path because more data is looked at.
- Storage device containing the data. The relative speed of the storage device containing the actual data and the device where the access path is stored has an effect on the time needed to rebuild an access path.
- The order of the records in the file. The system tries to rebuild an access path so that it can find information quickly when using that access path. The order of the records in a file has a small impact on how fast the system can build the access path while trying to maintain an efficient access path.

All of the preceding factors must be considered when estimating the amount of time to rebuild an access path.

Journaling Access Paths

Journaling access paths can significantly reduce recovery time by reducing the number of access paths that need to be rebuilt after an abnormal system end.

When you journal data base files, images of changes to the records in the file are recorded in the journal. These record images are used to recover the file should the system end abnormally. However, after an abnormal end, the system may find that access paths built over the file are not synchronized with the data in the file. If an access path and its data are not synchronized, the system must rebuild the access path to ensure that the two are synchronized and usable.

When access paths are journaled, the system records images of the access path in the journal to provide known synchronization points between the access path and its data. By having that information in the journal, the system can recover both the data files and the access paths, and ensure that the two are synchronized. In such cases, the lengthy time to rebuild the access paths can be avoided.

In addition, journaling access paths works with other recovery functions on the system. For example, the system has a number of options to help reduce the time required to recover from the failure and replacement of a disk device. These options include user auxiliary storage pools and checksum protection. While these options reduce the chances that the entire system must be reloaded because of the disk failure, they do not change the fact that access paths may need to be rebuilt when the system is started following replacement of the failed disk. By using access path journaling and some of the recovery options discussed previously, you can reduce your chances of having to reload the entire system and having to rebuild access paths.

Journaling access paths is easy to start. The Start Journal Access Path (STRJRNAP) command is used to start journaling the access path for the specified file. You can journal access paths that have a maintenance attribute of immediate (*IMMED) or delayed (*DLY). Once journaling is started, the system continues to protect the access path until the access path is deleted or you run the End Journaling Access Path (ENDJRNAP) command for that access path.

Before journaling an access path, you must start journaling for the physical files associated with the access path. In addition, you must use the same journal for the access path and its associated physical files.

Access path journaling is designed to minimize additional output operations. For example, the system will write the journal data for the changed record and the changed access path in the same output operation. However, you should seriously consider isolating your journals and journal receivers in user auxiliary storage pools when you start journaling your access paths. Placing these objects in their own user auxiliary storage pool will provide the best journaling performance, while helping to protect them from a disk failure.

For more information on journaling access paths, see the *Backup and Recovery Guide*.

Other Methods to Avoid Rebuilding Access Paths

If you do not journal your access paths, then you might consider some other system functions that can help you reduce the chances of having to rebuild access paths.

The method used by the system to determine if an access path needs to be rebuilt is a file synchronization indicator. Normally the synchronization indicator is on, indicating that the access path and its associated data are synchronized. When a job changes a file that affects an access path, the system turns off the synchronization indicator in the file. If the system ends abnormally, it must rebuild any access path whose file has its synchronization indicator off.

To reduce the number of access paths that must be rebuilt, you need a way to periodically synchronize the data with its access path. There are several methods to synchronize a file with its access path:

- Full file close. The last full (that is, not shared) system-wide close performed against a file will synchronize the access path and the data.
- Force access path. The force-access-path (FRCACCPTH) parameter can be specified on the create or change file commands.
- Force write ratio of 2 or greater. The force-write-ratio (FRCRATIO) parameter can be specified on the create, change, or override data base file commands.
- Force end of data. The file's data and its access path can be synchronized by running the force-end-of-data operation in your program. (Some high-level languages do not have a force-end-of-data operation. See your high-level language guide for further details.)

Keep in mind that while the data and its access path are synchronized after performing one of the methods mentioned previously, the next change to the data in the file can cause the synchronization indicator to be turned off again. It is also important to note that each of the methods can be costly in terms of performance; therefore, they should be used with caution. Consider journaling access paths, along with saving access paths, as the primary means of protecting access paths.

Controlling When Access Paths Are Rebuilt

If the system ends abnormally, during the next IPL the system automatically lists those files requiring access path recovery. You can decide whether to rebuild the access path:

- During the IPL
- After the IPL
- When the file is first used

The access path recovery value for a file is determined by the value you specified on the RECOVER parameter on the create and change file commands. You can override this value on the IPL Access Path Recovery display.

For those files that do not have to be rebuilt during IPL time, specify that the file can be rebuilt at a later time. This should help reduce the number of files that need to be rebuilt at IPL, allowing the system to complete its IPL much faster.

For example, you can specify that all files that must have their access path rebuilt should rebuild the access path when the file is first used. In this case, no access paths are rebuilt at IPL. You can control the order in which the access paths are rebuilt by running only those programs that use the files you want rebuilt first. This method will shorten the IPL time (because there are no access paths to rebuild during the IPL) and get the first several applications available faster. Note, however, that the overall time to rebuild all the access paths will probably be longer (because there may be other work running concurrently with rebuilding the access path and there may be less main storage available to rebuild the access path).

Designing Files to Reduce Access Path Rebuilding Time

File design can also help reduce access path recovery time. For example, you might divide a large master file into a history file and a transaction file. The transaction file would be used for adding new data, the history file would be used for inquiry only. On a daily basis, you might merge the transaction data into the history file, then clear the transaction file for the next day's data. With this design, the time to rebuild access paths could be shortened. That is, if the system abnormally ended during the day the access path to the smaller transaction file might need to be rebuilt. However, the access path to the large history file, being read-only for most of the day, would rarely be out of synchronization with its data, thereby significantly reducing the chances that it would have to be rebuilt.

Consider the trade-off between using a file design to reduce access path rebuilding time and using system-supplied functions like access path journaling. The file design described above may require a more complex application design. After evaluating your situation, you may decide to use system-supplied functions like access path journaling rather than design more complex applications.

Data Base Recovery after an Abnormal System End

After an abnormal system end, the system proceeds through several automatic recovery steps. This includes such things as: rebuilding the system directory and synchronizing the journal to the files being journaled. The system performs recovery operations during IPL and after IPL.

Data Base File Recovery during the IPL

During IPL, nothing but the recovery function is active on the system. During IPL, data base file recovery consists of the following:

- The following functions that were in progress when the system ended are completed:
  - Delete file
  - Remove member
  - Rename member
  - Move object
  - Rename object
  - Change object owner
  - Change file
  - Change member
  - Grant authority
  - Revoke authority
  - Start journaling physical file
  - Start journaling access path
  - End journaling physical file
  - End journaling access path
  - Recover SQL/400 views
- The following functions that were in progress when the system ended are backed out (you must run them again):
  - Create file
  - Add member
- If the operator is doing the IPL, the Override Access Path Recovery display appears on the operator's display. The display gives the operator the ability to override the RECOVER option for the files with immediate or delayed mainte-

nance which were in use. If no files were in use, or the IPL is unattended, no displays appear.

- Access paths that have immediate or delayed maintenance, that were in use, and that are specified for recovery during IPL (from the RECOVER option or overridden from the Override Access Path Recovery display) are rebuilt and a message is sent to the system operator. Files with journaled access paths that were in use are not displayed on the Override Access Path Recovery display. They are automatically recovered and a message is sent to the system operator.
- For unattended IPLs, if the system value QDBRCVYWT is 1 (wait), files that were in use that are specified for recovery after IPL are treated as files specified for recovery during IPL. See the *Work Management Guide* for more information on the system value QDBRCVYWT.
- Messages about the following information are sent to the history log:
  - The success or failure of the previously listed items
  - The physical file members that were open when the system ended abnormally and the last active relative record number in each member
  - The files that were open but whose access paths were not rebuilt because RECOVER(*NO) on the create or change file commands was specified for the files
  - The physical file members that could not be synchronized with the journal
  - That IPL data base recovery has completed
- The files whose access paths are to be rebuilt after IPL are allocated (see Figure 16-1 on page 16-10).

Data Base File Recovery after the IPL

This recovery step is run after the IPL is complete. Interactive users may be active and batch jobs running concurrently with this step of data base recovery. However, a file requiring recovery during this step is locked and not available to other jobs until recovery is completed on the access path.

Recovery after the IPL consists of the following:

- The access paths for immediate or delayed maintenance files which specify recovery after IPL, are rebuilt (see Figure 16-1 on page 16-10). A message is sent to the system operator indicating the number of access paths that are to be rebuilt. As each access path is rebuilt, its associated file is unlocked. (If a program tries to use a file that is being rebuilt, the program must wait until the rebuilding is complete.)
- Messages are sent to the system history log indicating the success or failure of the rebuild operation and that data base recovery is complete.

Note: If you are not using journaling for a file and you were adding records to a member when the system ended abnormally, the records are in sequence after recovery. That is, if you added nine records and record 9 is there after recovery, records 1 through 8 are also there. However, if you were updating or deleting records when the system ended abnormally and you were not journaling, not all records you were updating are necessarily updated after recovery and not all records you were deleting are necessarily deleted after recovery.

Data Base File Recovery Options Table

The table below summarizes the file recovery options:

	RECOVER Parameter Specified		
Access Path/ Maintenance	***NO**	***AFTIPL**	***IPL**
Keyed sequence access path/ immediate or delayed maintenance	• No data base recovery at IPL • File available immediately • Access path rebuilt first time file opened	• Access path rebuilt after IPL • File allocated • For logical files, the underlying physical data spaces are locked for shared read and are available immediately	• Access path rebuilt during IPL
Keyed sequence access path rebuild maintenance	• No data base recovery at IPL • File available immediately • Access path rebuilt first time file opened	• Not applicable; no recovery is done for rebuild maintenance	• Not applicable; no recovery is done for rebuild maintenance
Arrival sequence access path	• No data base recovery at IPL • File available immediately	• Not applicable; no recovery is done for an arrival sequence access path	• Not applicable; no recovery is done for an arrival sequence access path

Figure 16-1. Relationship of Access Path, Maintenance, and Recovery

Chapter 17. Using Source Files

This chapter describes source files. Source file concepts are discussed, along with why you would use a source file. In addition, this chapter describes how to set up a source file, how to enter data into a source file, and how to use that source file to create another object (for example, a file or program) on the system.

Source File Concepts

A source file is used when a command alone cannot supply sufficient information for creating an object. It contains input (source) data needed to create some types of objects. For example, to create a control language (CL) program, you must use a source file containing source statements, which are in the form of commands. To create a logical file, you must use a source file containing DDS.

To create the following objects, source files are required:

- High-level language programs
- Control language programs
- Logical files
- Intersystem communications function files
- Commands
- Translate tables

To create the following objects, source files can be used, but are *not* required:

- Physical files
- Display files
- Printer files

A source file can be a data base file, diskette file, tape file, or inline data file. (An inline data file is included as part of a job.) A source data base file is simply another type of data base file. You can use a source data base file like you would any other data base file on the system.

Creating a Source File

To create a source file, you can use the Create Source Physical File (CRTSRCPF), Create Physical File (CRTPF), or Create Logical File (CRTLF) command. Normally, you will use the CRTSRCPF command to create a source file, because many of the parameters default to values that you usually want for a source file. (If you want to use DDS to create a source file, then you would use the CRTPF or CRTLF command.)

The CRTSRCPF command creates a physical file, but with attributes appropriate for source physical files. For example, the default record length for a source file is 92 (80 for the source data field, 6 for the source sequence number field, and 6 for the source date field).

The following example shows how to create a source file using the CRTSRCPF command and using the command defaults:

```
CRTSRCPF FILE(QGPL/FRSOURCE) TEXT('Source file')
```

IBM-Supplied Source Files

For your convenience, OS/400 and other licensed programs provide a data base source file for each type of source. These source files are:

File Name	Library Name	Used to Create
QBASSRC	QBAS	BASIC programs
QCLSRC	QGPL	CL programs
QCMDSRC	QGPL	Command definition statements
QDDSSRC	QGPL	Files
QFMTSRC	QGPL	Sort source
QLBLSRC	QLBL	COBOL/400 programs
QPLISRC	QPLI	PL/I programs
QRPGSRC	QRPG	RPG/400 programs
QPASSRC	QPAS	Pascal programs
QTBLSRC	QGPL	Translate tables
QTXTSRC	QPDA	Text

You can either add your source members to these files or create your own source files. Normally, you will want to create your own source files using the same names as the IBM-supplied files, but in different libraries (IBM-supplied files may get overlaid when a new release of the system is installed). The IBM-supplied source files are created with the file names used for the corresponding create command (for example, the CRTCLPGM command uses the QCLSRC file name as the default). Additionally, the IBM-supplied programmer menu uses the same default names. If you create your own source files, do not place them in the same library as the IBM-supplied source files. (If you use the same file names as the IBM-supplied names, you should ensure that the library containing your source files precedes the library containing the IBM-supplied source files in the library list.)

Source File Attributes

Source files usually have the following attributes:

- A record length of 92 characters (this includes a 6-byte sequence number, a 6-byte date, and 80 bytes of source).
- Keys (sequence numbers) that are unique even though the access path does not specify unique keys. You are not required to specify a key for a source file. Default source files are created without keys (arrival sequence access path). A source file created with an arrival sequence access path requires less storage space and reduces save/restore time in comparison to a source file for which a keyed sequence access path is specified.
- More than one member.
- Member names that are the same as the names of the objects that are created using them.
- The same record format for all records.
- Relatively few records in each member compared to most data files.

Some restrictions are:

- The source sequence number must be used as a key, if a key is specified.
- The key, if one is specified, must be in ascending sequence.
- The access path cannot specify unique keys.
- The ALTSEQ keyword is not allowed in DDS for source files.
- The first field must be a 6-digit sequence number field containing zoned decimal data and zero decimal digits.

- The second field must be a 6-digit date field containing zoned decimal data and zero decimal digits.
- All fields following the second field must be zoned decimal or character.

Creating Source File without DDS

When you create a physical data base source file without using DDS, but by specifying the record length (RCDLEN parameter), the source created contains three fields: *SRCSEQ*, *SRCDAT*, and *SRCDTA*. (The record length must include 12 characters for sequence number and date-of-last-change fields so that the length of the data portion of the record equals the record length minus 12.) The data portion of the record can be defined to contain more than one field (each of which must be character or zoned decimal). If you want to define the data portion of the record as containing more than one field, you must define the fields using DDS.

A record format consisting of the following three fields is automatically used for a source physical file created using the Create Source Physical File (CRTSRCPF) command:

Field	Name	Data Type and Length	Description
1	SRCSEQ	Zoned decimal, 6 digits, 2 decimal positions	Sequence number for record
2	SRCDAT	Zoned decimal, 6 digits, no decimal positions	Date of last update of record
3	SRCDTA	Character, any length	Data portion of the record (text)

Note: For all IBM-supplied data base source files, the length of the data portion is 80 bytes. For IBM-supplied device source files, the length of the data portion is the maximum record length for the associated device.

Creating Source Files with DDS

If you want to create a source file for which you need to define the record format, use the Create Physical File (CRTPF) or Create Logical File (CRTLF) command. If you create a source logical file, the logical file member should only refer to one physical file member to avoid duplicate keys.

Working with Source Files

The following section describes how to enter and maintain data in your source files.

Using the Source Entry Utility

You can use SEU, to enter and change source in a source file. If you use SEU to enter source in a data base file, SEU adds the sequence number and date fields to each source record. (See the *SEU User's Guide/Reference* for more information about SEU.)

If you use SEU to update a source file, you can add records between existing records. For example, if you add a record between records 0003.00 and 0004.00, the sequence

number of the added record could be 0003.01. SEU will automatically arrange the newly added statements in this way.

When records are first placed in a source file, the date field is all zoned decimal zeros (unless DDS is used with the DFT keyword specified). If you use SEU, the date field changes in a record when you change the record.

Using Device Source Files

Tape and diskette device files can be created as source files. When device files are used as source files, the record length must include the sequence number and date fields. Any maximum record length restrictions must consider these additional 12 characters. For example, the maximum record length for a tape record is 32,766. If data is to be processed as source input, the actual tape data record has a maximum length of 32,754 (which is 32,766 minus 12).

If you open source device files for input, the system adds the sequence number and date fields, but there are zeros in the date fields.

If you open a device file for output and the file is defined as a source file, the system deletes the sequence number and date before writing the data to the device.

Copying Source File Data

The Copy Source File (CPYSRCF) and Copy File (CPYF) commands can be used to write data to and from source file members.

Using the Copy Source File (CPYSRCF) Command for Copying to and from Source Files

The CPYSRCF command is designed to operate with data base source files. Although it is similar, in function, to the Copy File (CPYF) command, the CPYSRCF command provides defaults that are normally used when copying a source file. For example, it has a default that assumes the TOMBR parameter is the same as the FROMMBR parameter and that any TOMBR records will always be replaced. The CPYSRCF command also supports a unique printing format when TOFILE(*PRINT) is specified. Therefore, when you are copying data base source files, you will probably want to use the CPYSRCF command.

Using the Copy File (CPYF) Command for Copying to and from Files

The CPYF command provides additional functions over the CPYSRCF command such as:

- Copying from data base source files to device files
- Copying from device files to data base files
- Copying between nonsource data base files and source data base files
- Printing a source member in hexadecimal format
- Copying source with selection values

See the *CL Reference* and the *Data Management Guide* for more information about the copy commands.

Source Sequence Numbers Used in Copies

When you copy to a data base source file, you can use the SRCOPT parameter to update sequence numbers and initialize dates to zeros. By default, the system assigns a sequence number of 1.00 to the first record and increases the sequence numbers by 1.00 for the remaining records. You can use the SRCSEQ parameter to set a fractional increment value and to specify the sequence number at which the renumbering is to start. For example, if you specify in the SRCSEQ parameter that the increment value is .10 and is to start at sequence number 100.00, the copied records have the sequence numbers 100.00, 100.10, 100.20, and so on.

If a starting value of .01 and an increment value of .01 are specified, the maximum number of records that can have unique sequence numbers is 999,999. When the maximum sequence number (9999.99) is reached, any remaining records will have a sequence number of 9999.99.

The following is an example of copying source from one member to another in the same file. If MBRB does not exist, it is added; if it does exist, all records are replaced.

```
CPYSRCF FROMFILE(QCLSRC) TOFILE(QCLSRC) FROMMBR(MBRA) +
        TOMBR(MBRB)
```

The following is an example of copying a generic member name from one file to another. All members starting with PAY are copied. If the corresponding members do not exist, they are added; if they do exist, all records are replaced.

```
CPYSRCF FROMFILE(LIB1/QCLSRC) TOFILE(LIB2/QCLSRC) +
        FROMMBR(PAY*)
```

The following is an example of copying the member PAY1 to the printer file QSYSPRT (the default for *PRINT). A format similar to the one used by SEU is used to print the source statements.

```
CPYSRCF FROMFILE(QCLSRC) TOFILE(*PRINT) FROMMBR(PAY1)
```

When you copy from a device source file to a data base source file, sequence numbers are added and dates are initialized to zeros. Sequence numbers start at 1.00 and are increased by 1.00. If the file being copied has more than 9999 records, then the sequence number is wrapped back to 1.00 and continues to be increased unless the SRCOPT and SRCSEQ parameters are specified.

When you are copying from a data base source file to a device source file, the date and sequence number fields are removed.

Using Source Files in a Program

You can process a source file in your program. You can use the external definition of the source file and do any input/output operations for that file, just as you would for any other data base file.

Source files are externally described data base files. As such, when you name a source file in your program and compile it, the source file description is automatically included in your program printout. For example, assume you wanted to read and update records for a member called FILEA in the source file QDDSSRC. When you write the program to process this file, the system will include the *SRCSEQ*, *SRCDAT*, and *SRCDTA* fields from the source file.

Note: You can display the fields defined in a file by using the Display File Field Description command (DSPFFD). For more information about this command,

see "Displaying the Descriptions of the Fields in a File" on page 15-1 and the DSPFFD command in the *CL Reference*.

The program processing the FILEA member of the QDDSSRC file could:

- Open the file member (just like any other data base file member).
- Read and update records from the source file (probably changing the *SRCDTA* field where the actual source data is stored).
- Close the source file member (just like any other data base file member).

Creating an Object Using a Source File

You can use a create command to create an object using a source file. If you create an object using a source file, you can specify the name of the source file on the create command.

For example, to create a CL program, you use the Create Control Language Program (CRTCLPGM) command. A create command specifies through a SRCFILE parameter where the source is stored.

The create commands are designed so that you do not have to specify source file name and member name if you do the following:

1. Use the default source file name for the type of object you are creating. (To find the default source file name for the command you are using, see "IBM-Supplied Source Files" on page 17-2.)
2. Give the source member the same name as the object to be created.

For example, to create the CL program PGMA using the command defaults, you would simply type:

```
CRTCLPGM PGM(PGMA)
```

The system would expect the source for PGMA to be in the PGMA member in the QCLSRC source file. The library containing the QCLSRC file would be determined by the library list.

As another example, the following Create Physical File (CRTPF) command creates the file DSTREF using the data base source file FRSOURCE. The source member is named DSTREF. Because the SRCMBR parameter is not specified, the system assumes that the member name, DSTREF, is the same as the name of the object being created.

```
CRTPF FILE(QGPL/DSTREF) SRCFILE(QGPL/FRSOURCE)
```

Creating an Object from Source Statements in a Batch Job

If your create command is contained in a batch job, you can use an inline data file as the source file for the command. However, inline data files used as a source file should not exceed 10,000 records. The inline data file can be either named or unnamed. Named inline data files have a unique file name that is specified on the //DATA command. (For more information about inline data files, see the *Data Management Guide*.

Unnamed inline data files are files without unique file names; they are all named QINLINE. The following is an example of an inline data file used as a source file:

```
//BCHJOB
CRTPF FILE(DSTPRODLB/ORD199) SRCFILE(QINLINE)
//DATA FILETYPE(*SRC)
  .
  .   (source statements)
  .
//
//ENDBCHJOB
```

In this example, no file name was specified on the //DATA command. An unnamed spooled file was created when the job was processed by the spooling reader. The CRTPF command must specify QINLINE as the source file name to access the unnamed file. The //DATA command also specifies that the inline file is a source file (*SRC specified for the FILETYPE parameter).

If you specify a file name on the //DATA command, you must specify the same name on the SRCFILE parameter on the CRTPF command. For example:

```
//BCHJOB
CRTPF FILE(DSTPRODLB/ORD199) SRCFILE(ORD199)
//DATA FILE(ORD199) FILETYPE(*SRC)
  .
  .   (source statements)
  .
//
//ENDBCHJOB
```

If a program uses an inline file, the system searches for the first inline file of the specified name. If that file cannot be found, the program uses the first file that is unnamed (QINLINE).

If you do not specify a source file name on a create command, an IBM-supplied source file is assumed to contain the needed source data. For example, if you are creating a CL program but you did not specify a source file name, the IBM-supplied source file QCLSRC is used. You must have placed the source data in QCLSRC.

If a source file is a data base file, you can specify a source member that contains the needed source data. If you do not specify a source member, the source data must be in a member that has the same name as the object being created.

Determining Which Source File Member Was Used to Create an Object

When an object is created from source, the information about the source file, library, and member is retained in the object. The date/time that the source member was last changed before object creation is also saved in the object.

The information in the object can be displayed with the Display Object Description (DSPOBJD) command and specifying DETAIL(*SERVICE).

This information can help you in determining which source member was used and if the existing source member was changed since the object was created.

You can also ensure that the source used to create an object is the same as the source that is currently in the source member using the following commands:

- The Display File Description (DSPFD) command using TYPE(*MBR). This display shows both date/times for the source member. The `Last source update date/time` value should be used to compare to the `Source file date/time` value displayed from the DSPOBJD command.
- The Display Object Description (DSPOBJD) command using DETAIL(*SERVICE). This display shows the date/time of the source member used to create the object.

Note: If you are using the data written to output files to determine if the source and object dates are the same, then you can compare the *ODSRCD* (source date) and *ODSRCT* (source time) fields from the output file of the DSPOBJD DETAIL(*SERVICE) command to the *MBUPDD* (member update date) and *MBUPDT* (member update time) fields from the output file of the DSPFD TYPE(*MBR) command.

Managing a Source File

This section describes several considerations for managing source files.

Changing a Source File's Attributes

If you are using SEU to maintain data base source files, see the *SEU User's Guide/Reference* for how to change data base source files. If you are not using SEU to maintain data base source files, you must totally replace the existing member.

If your source file is on a diskette, you can copy it to a data base file, change it using SEU, and copy it back to a diskette. If you do not use SEU, you have to delete the old source file and create a new source file.

If you change a source file, the object previously created from the source file does not match the current source. The old object must be deleted, and a create command must be specified to re-create the object using the changed source file. For example, if you change the source file FRSOURCE created in "Creating an Object Using a Source File" on page 17-6, you have to delete the file DSTREF that was created from the original source file, and create it again using the new source file so that DSTREF matches the changed FRSOURCE source file.

Reorganizing Source File Member Data

You usually do not need to reorganize a source file member if you use arrival sequence source files.

To assign unique sequence numbers to all the records, specify the following parameters on the Reorganize Physical File Member (RGZPFM) command:

- KEYFILE(*NONE), so that the records are not reorganized
- SRCOPT(*SEQNBR), so that the sequence numbers are changed
- SRCSEQ with a fractional value such as .10 or .01, so that all the sequence numbers are unique

Note: Deleted records, if they exist, will be compressed out.

A source file with an arrival sequence access path can be reorganized by sequence number if a logical file for which a keyed sequence access path is specified is created over the physical file.

Determining When a Source Statement Was Changed

Each source record contains a date field which is automatically updated by SEU if a change occurs to the statement. This can be used to determine when a statement was last changed. Most high-level language compilers print these dates on the compiler listings. The Copy File (CPYF) and Copy Source File (CPYSRCF) commands also print these dates.

Each source member description contains two date and time fields. The first date/time field reflects changes caused by SEU any time the member is closed after it was updated.

The second date/time field reflects any changes to the member. This includes all changes caused by SEU, the CPYSRCF command, authorization changes, and changes to the file status. For example, the FRCRATIO parameter on the Change Physical File (CHGPF) command changes the member status. This date/time is used by the Save Changed Objects (SAVCHGOBJ) command to determine if the member should be saved. Both date/times can be displayed with the Display File Description (DSPFD) command specifying TYPE(*MBR). There are two change date/times shown with source members:

- `Last source update date/time`. This value reflects any change to the source data records in the member. When a source update occurs, the `Last change date/time` value is also updated although there may be a 1- or 2-second difference in that time/date value.
- `Last change date/time`. This value reflects any changes to the member. This includes all changes caused by SEU, CPYSRCF, authorization changes, or changes to file status. For example, the FRCRATIO parameter on the CHGPF command changes the member status, and therefore, is reflected in the `Last change date/time` value.

Considerations for Saving Source Files

When you save an object to tape or diskette, the system updates the object description with the date and time of the save operation. When you save an object to a save file, you can prevent the system from updating the date and time of the save operation by specifying UPDHST(*NO) on the save command. When you restore an object, the system always updates the object description with the date and time of the restore operation. You can display this and other save/restore related information by using the Display Object Description (DSPOBJD) command with DETAIL(*FULL). Use the Display Save File (DSPSAVF) command to display the objects in a save file.

Specify DATA(*SAVRST) on the Display Diskette (DSPDKT) or Display Tape (DSPTAP) command for a display of the objects on the media.

The last save/restore date for data base members can be displayed by typing:

```
DSPFD FILE(file-name) TYPE(*MBR)
```

For more information about saving and restoring data base files (including source files), see the *Backup and Recovery Guide*.

Using Source Files for Documentation

You can use the IBM-supplied source file QTXTSRC to help you create and update online documentation.

You can create and update QTXTSRC members just like any other application (such as QRPGSRC or QCLSRC) available with SEU. The QTXTSRC file is most useful for narrative documentation, which can be retrieved online or printed. The text that you put in a source member is easy to update by using the SEU add, change, move, copy, and include operations. The entire member can be printed by specifying Yes for the print current source file option on the exit prompt. You can also write a program to print all or part of a source member.

Appendix A. Data Base File Sizes

The following data base file maximums should be kept in mind when designing files on the AS/400 system:

Description	Maximum Value
Number of bytes in a record	32,766 bytes
Number of fields in a record format	8,000 fields
Number of key fields in a file	120 fields
Size of key for physical and logical files	120 characters
Size of key for ORDER BY (SQL/400 and OPNQRYF)	256 characters
Number of records contained in a file	16,777,215 records[1]
Number of bytes in a file	2,147,483,648 bytes
Number of bytes in an access path	1,073,741,824 bytes
Number of keyed logical files built over a physical file	3,686 files
Number of physical file members in a logical file member	32 members
Number of members that can be joined	32 members
Size of a character or DBCS field	32,766 characters
Size of a zoned decimal or packed decimal field	31 digits

[1] For files with keyed sequence access paths, the maximum number of records in a member varies and can be estimated using the following formula:

$$\frac{716{,}800{,}000}{10 + (.8 * \text{key length})}$$

This is an estimated value, the actual maximum number of records may vary significantly from the number determined by this formula.

These are maximum values. There are situations where the actual limit you experience will be less than the stated maximum. For example, certain high-level languages may have more restrictive limits than those described above.

Keep in mind that performance may suffer as you approach some of these maximums. For example, the more logical files you have built over a physical file, the greater the chance that system performance may suffer (if you are frequently changing data in the physical file that causes a change in many logical file access paths).

Normally, an AS/400 data base file can grow until it reaches the maximum size allowed on the system. The system normally will not allocate all the file space at once. Rather, the system will occasionally allocate additional space as the file grows larger. This method of automatic storage allocation provides the best combination of good performance and effective auxiliary storage space management.

If you want to control the size of the file, the storage allocation, and whether the file should be contiguous on auxiliary storage, you can use the SIZE, ALLOCATE, and CONTIG parameters on the Create Physical File (CRTPF) and the Create Source Physical File (CRTSRCPF) commands. For more information about how to specify these parameters, see the *CL Reference*.

You can use the following formulas to estimate the disk size of your physical and logical files.

- For a physical file (excluding the access path):

  Disk size = (number of valid and deleted records + 1) * (record length + 1) + 5120 * (number of members) + 1024

  The size of the physical file depends on the SIZE and ALLOCATE parameters on the CRTPF and CRTSRCPF commands. If you specify ALLOCATE(*YES), the initial allocation and increment size on the SIZE keyword must be used instead of the number of records.
- For a logical file (excluding the access path):

  Disk size = (2560) * (number of members) + 1024

- For a keyed sequence access path, per member:

  (number of keys) * (key length + 8) * (0.8) * (1.85) + 4096

  Notes:
  1. The number of keys is the number of index entries in the access path.
  2. The key length is the sum of the length of all the key fields.
  3. 0.8 is an estimate of key compression.
  4. 1.85 is an estimate of index overhead and space reserved for inserting keys.

This estimate may differ significantly from your file. The keyed sequence access path depends heavily on the data in your records. The only way to get an accurate size is to load your data and display the file description.

The following is a list of minimum file sizes:

Description	Minimum Size
Physical file without a member	1024 bytes
Physical file with a single member	5632 bytes
Keyed sequence access path	1024 bytes

Note: No additional space is required for an arrival sequence access path.

In addition to the file sizes, the system maintains internal formats and directories for data base files. (These internal objects are owned by user profile QDBSHR.) The following are estimates of the sizes of those objects:

- For any file not sharing another file's format:

  Format size = (75 * number of fields) + 512

- For files sharing their format with any other file:

  Format sharing directory size = (16 * number of files sharing the format) + 272

- For each physical file and each physical file member having a logical file or logical file member built over it:

  Data sharing directory size = (16 * number of files or members sharing data) + 272

- For each file member having a logical file member sharing its access path:

  Access path sharing directory size = (16 * number of files or members sharing access path) + 272

Appendix B. Double-Byte Character Set (DBCS) Considerations

The double-byte character set (DBCS) is a character set that represents each character with 2 bytes. The DBCS supports national languages that contain a large number of unique characters or symbols (the maximum number of characters that can be represented with 1 byte is 256 characters). Examples of such languages include Japanese, Korean, and Chinese.

This appendix describes DBCS considerations as they apply to the data base on the AS/400 system.

DBCS Field Data Types

The DBCS data types (specified in position 35 on the *DDS Coding Form*) are:

Entry	Meaning
J	DBCS-only
O	DBCS-open
E	DBCS-either

A DBCS-open field is a character string that contains both single- and double-byte characters. A DBCS-only field is a character string that contains a shift-out character, followed by double-byte characters, followed by a shift-in character. A DBCS-either field is a character string that contains either all single-byte data or all double-byte data.

DBCS Field Mapping Considerations

The following chart shows what types of data mapping are valid between physical and logical files for DBCS fields:

Physical File Data Type	Logical File Data Type			
	Character	**DBCS-open**	**DBCS-either**	**DBCS-only**
Character	Valid	Valid	Valid	Not valid
DBCS-open	Not valid	Valid	Not valid	Not valid
DBCS-either	Not valid	Valid	Valid	Not valid
DBCS-only	Not valid	Valid	Valid	Valid

DBCS Field Concatenation

When fields are concatenated, the data types can change (the resulting data type is automatically determined by the system). The following shows what data types result:

	Any Numeric	Character	DBCS-open	DBCS-either	DBCS-only
Any Numeric	Zoned decimal	Character	DBCS-open	DBCS-open	DBCS-open
Character	Character	Character	DBCS-open	DBCS-open	DBCS-open
DBCS-open	DBCS-open	DBCS-open	DBCS-open	DBCS-open	DBCS-open
DBCS-either	DBCS-open	DBCS-open	DBCS-open	DBCS-open	DBCS-open
DBCS-only	DBCS-open	DBCS-open	DBCS-open	DBCS-open	DBCS-only

Notes:

1. When DBCS fields are concatenated, a shift-out at the end of one field and a shift-in at the end of the next field are eliminated.
2. A concatenated field that contains DBCS fields must be an input-only field.

Using DBCS Fields in the Open Query File (OPNQRYF) Command

This section describes considerations when using DBCS fields in the Open Query File (OPNQRYF) command.

Comparing DBCS Fields

When comparing two fields or constants, the data types must be compatible. The following table describes valid comparisons for DBCS fields:

	Any Numeric	Character	DBCS-open	DBCS-either	DBCS-only
Any Numeric	Valid	Not valid	Not valid	Not valid	Not valid
Character	Not valid	Valid	Valid	Valid	Valid
DBCS-open	Not valid	Valid	Valid	Valid	Valid
DBCS-either	Not valid	Valid	Valid	Valid	Valid
DBCS-only	Not valid	Not valid	Valid	Valid	Valid

Using the Wild Card Function with DBCS Fields

When using the "wild card" (%WLDCRD) function with any DBCS field, both single-byte and double-byte wild card values (asterisk and underline) are allowed. The following special rules apply:

- A single-byte underline refers to one EBCDIC character; a double-byte underline refers to one double-byte character.
- A single- or double-byte asterisk refers to any number of characters of any type.

Appendix C. Data Base Lock Considerations

The following table summarizes some of the most commonly used data base functions and the types of locks they place on data base files.

Function	Command	File Lock	Member/Data Lock	Access Path Lock
Add Member	ADDPFM, ADDLFM	LEAR		LEAR
Change File Attributes	CHGPF,CHGLF	LENR	LEAR	LEAR
Change Member Attributes	CHGPFM, CHGLFM	LSRD	LEAR	
Check Object	CHKOBJ	LSRO		
Clear Physical File Member	CLRPFM	LSRO	LENR	
Create Duplicate Object	CRTDUPOBJ	LENR		
Create File	CRTPF, CRTLF, CRTSRCPF	LENR		
Delete File	DLTF	LENR		LEAR
Grant/Revoke Authority	GRTOBJAUT, RVKOBJAUT	LENR		
Initialize Physical File Member	INZPFM	LSRD	LEAR	
Move Object	MOVOBJ	LENR		
Open File		LSRD	LSRD	LEAR
Rebuild Access Path		LSRD	LSRD	LENR
Remove Member	RMVM	LEAR	LENR	LEAR
Rename File	RNMOBJ	LENR	LENR	LENR
Rename Member	RNMM	LEAR	LENR	LENR
Reorganize Physical File Member	RGZPFM	LENR		
Restore File	RSTLIB, RSTOBJ	LENR		
Save File	SAVLIB, SAVOBJ, SAVCHGOBJ	LSRO	LSRO	

The following table shows the valid lock combinations:

Lock	LENR	LEAR	LSUP	LSRO	LSRD
LENR[1]					
LEAR[2]					X
LSUP[3]			X		X
LSRO[4]				X	X
LSRD[5]		X	X	X	X

1 Exclusive lock. The object is allocated for the exclusive use of the requesting job; no other job can use the object.

2 Exclusive lock, allow read. The object is allocated to the job that requested it, but other jobs can read the object.

3 Shared lock, allow read and update. The object can be shared either for read or change with other jobs.

4 Shared lock, read only. The object can be shared for read with other jobs.

5 Shared lock. The object can be shared with another job if the job does not request exclusive use of the object.

Appendix D. Office Output File Considerations

This appendix discusses output files for the AS/400 Office commands. The terms outfile and output file are used interchangeably.

Display Directory (DSPDIR) Outfile Considerations

The Display Directory command can be used to create a data base file that contains the information from the system distribution directory. This outfile will contain either the basic or full information about the directory entries depending on the USRID, USER, and DETAIL parameters.

If the USRID or USER parameters specified a unique entry, one detail data base record is written with the full directory details. If more than one description exists for the specified USRID or USER, the outfile will contain a full detail record for each description.

When the USRID specifies, or defaults to *ALL, the DETAIL parameter will determine what data will be written to the data base file. If DETAIL(*BASIC) was specified, the data base file will only contain the user ID, address, and description. If DETAIL(*FULL) was specified, a full detail record is built for each description. Note that when multiple descriptions exist in the directory for a given user ID and address, one record will be written for each description. The only data which is different between these records is the description field.

The model outfile supplied with the system is named QAOSDIRO in library QSYS.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QAOSDIRO in library QSYS
A*  File Type:  Physical
A*  Record Format Name:  OSDIRE
A*
A*
A          R OSDIRE
A            WOSDDEN        8A         TEXT('User ID')
A            WOSDDGN        8A         TEXT('Address')
A            WOSDDESC      50A         TEXT('User description')
A            WOSDREN        8A         TEXT('System name')
A            WOSDRGN        8A         TEXT('System group')
A            WOSDUSRP      10A         TEXT('User profile')
A            WOSDINDU       1A         TEXT('Indirect user flag')
A            WOSDRCVP       1A         TEXT('Print personal mail')
A            WOSDADD1      35A         TEXT('Mailing address 1')
A            WOSDADD2      35A         TEXT('Mailing address 2')
A            WOSDADD3      35A         TEXT('Mailing address 3')
A            WOSDADD4      35A         TEXT('Mailing address 4')
A            WOSDLOC       35A         TEXT('Location')
A            WOSDTEL1      25A         TEXT('Telephone number 1')
A            WOSDTEL2      25A         TEXT('Telephone number 2')
A            WOSDTEXT      50A         TEXT('Misc. text')
A*
A* End of specifications
```

The meaning of each field in the record is:

- *WOSDDEN* - user ID. The user identification for which the detailed information is provided. This field is put into code from code page 256/character set 930. The user ID along with the address uniquely identifies the directory entry.
- *WOSDDGN* - address. The address for which the detailed information is provided. This field is put into code from code page 256/character set 930. The address along with the user ID uniquely identifies the directory entry.
- *WOSDDESC* - user description. The user description associated with the user ID and address.
- *WOSDREN* - system name. The system name (known to SNADS) of the system on which this user works. This field is put into code from code page 256/character set 930. The system name along with the system group uniquely identifies a system.
- *WOSDRGN* - system group. The system group (known to SNADS) of the system on which this user works. This field is put into code from code page 256/character set 930. The system group may be blank. The system group along with the system name uniquely identifies a system.
- *WOSDUSRP* - user profile. The system profile name for the user.
- *WOSDINDU* - indirect user flag. A one-character flag to indicate whether or not the user is an indirect user (one who does not use the system to receive mail, but receives printed mail). The value 1 indicates the user is an indirect user. The value 0 indicates that the user is not an indirect user.
- *WOSDRCVP* - receive personal mail. A one-character flag to specify whether an indirect user is to have personal mail printed. The value 1 indicates the indirect user is to receive personal mail. The value 0 indicates that no personal mail will be printed for the indirect user. The data in this field has no meaning if this is not an indirect user.
- *WOSDADD1* - address line 1. The first address line for the user. This field is optional input to the directory and may contain blanks.
- *WOSDADD2* - address line 2. The second address line for the user. This field is optional input to the directory and may contain blanks.
- *WOSDADD3* - address line 3. The third address line for the user. This field is optional input to the directory and may contain blanks.
- *WOSDADD4* - address line 4. The fourth address line for the user. This field is optional input to the directory and may contain blanks.
- *WOSDLOC* - location. The location of the user. This may specify an item such as a site or department. This field is optional input to the directory and may contain blanks.
- *WOSDTEL1* - telephone number 1. The telephone number of the user. This field is optional input to the directory and may contain blanks.
- *WOSDTEL2* - telephone number 2. The second or alternative telephone number of the user. This field is optional input to the directory and may contain blanks.
- *WOSDTEXT* - text. Any additional information about the user may be contained in this field. It is optional input to the directory and may contain blanks.

Display Distribution List (DSPDSTL) Outfile Considerations

The Display Distribution List command can be used to create a data base file that contains the information about distribution lists. The data base file contents are determined by the LSTID and DETAIL parameters.

If the LSTID parameter specified a list ID, the entries within that list are written to the data base file. If the LSTID parameter specified *ALL and the DETAIL parameter specified *BASIC, each list ID and description within the directory is written to the data base file. When LSTID(*ALL) is specified and DETAIL(*FULL) is specified, each entry within each list is written to the data base file.

The data base file will contain a header record for each distribution list being written. This header record contains the list ID and description for the list being output. The header record may be followed by detail records which contain the user ID/address and description for each of the entries contained in the distribution list.

When the LSTID parameter specified a list ID, a header record is written for the list ID and description. It will be followed by a detail record for each entry in the list.

When the LSTID parameter specified or defaulted to *ALL and the DETAIL parameter specified *FULL, a header record is written for the first distribution list contained in the directory, followed by detail records for all entries in the list. A header record is then written for the next distribution list, again followed by detail records for all entries in that list. This continues until all distribution lists have been processed.

When the LSTID parameter specified or defaulted to *ALL and the DETAIL parameter specified *BASIC, only a header record is written for each distribution list contained in the directory.

The model outfile supplied with the system is named QAOSDSTO in library QSYS.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QAOSDSTO in library QSYS
A*  File Type:  Physical
A*  Record Format Name:  OSDSTL
A*
A*
A          R OSDSTL
A            WOSLLST             16A         TEXT('List ID')
A            WOSLRC               1A         TEXT('Record code')
A            WOSLEID             16A         TEXT('Entry ID')
A            WOSLDESC            50A         TEXT('Description')
A*
A* End of specifications
```

The meaning of each field in the record is:

- *WOSLLST* - list ID. The list ID of a distribution list contained in the directory. This is a two-part field, each part has 1-8 characters. This field is put into code from code page 256/character set 930.
- *WOSLRC* - record code. Defines the type of record:
  - 0 - list header record
  - 1 - list entry record

- *WOSLEID* - entry ID. This is the user ID and address of an entry contained in the distribution list when the record code is 1 (detail record). For record code 0 (header record), this field contains the local system name and system group on which this distribution list exists. This field is put into code from code page 256/character set 930.
- *WOSLDESC* - entry description. This field contains the description associated with the entry ID. When the record code is 0 (header record), the entry description contains the list description. When the record code is 1 (detail record), the entry description contains the user description of the entry in the list. If the user ID and address is in fact a remote distribution list, the entry description contains the remote list description.

Display Folder (DSPFLR) Outfile Considerations

The Display Folder command can be used to create a data base file that contains information about the documents within a folder or about the folders that are within a folder.

Folder Object Information

The model outfile supplied with the system is named QADSPFLR in library QSYS and is used when DSPFLR TYPE(*FLR) OUTPUT(*OUTFILE) is specified.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QADSPFLR in library QSYS
A*  File Type:  Physical
A*  Record Format Name:  FLRDTL
A*
A*
A          R FLRDTL
A            BASFLR              63A         TEXT('Folder path')
A            FLRNAM              12A         TEXT('Folder name')
A            DESC                44A         TEXT('Description')
A            LASTREV              8A         TEXT('Last revised')
A*
A* End of specifications
```

The meaning of each field in the record is:

- *BASFLR* - folder path name. The folder path name used to locate the folder in the *FLRNAM* field.
- *FLRNAM* - folder name. The name of each folder within the folder that is defined by the folder path name.
- *DESC* - folder description. The description associated with each folder. This may contain blanks if no description was provided when the folder was created.
- *LASTREV* - folder last revision date. This is the date the folder was last changed by either changing the folder description or by adding an object to or deleting an object from the folder. It is in the format YYYYMMDD where:

  YYYY = year
  MM = month
  DD = day

Document Object Information

The other model outfile supplied with the system is named QADSPDOC in library QSYS and is used when DSPFLR TYPE(*DOC) OUTPUT(*OUTFILE) is specified.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QADSPDOC in library QSYS
A*  File Type:  Physical
A*  Record Format Name:  DOCDTL
A*
A*
A          R DOCDTL
A            FLRNAM              63A         TEXT('Folder')
A            DOCNAM              12A         TEXT('Document')
A            DESC                44A         TEXT('Description')
A            LASTREV              8A         TEXT('Last revised')
A            DOCTYPE             10A         TEXT('Document type')
A*
A* End of specifications
```

The meaning of each field in the record is:

- *FLRNAM* - folder path name. The folder path name used to locate the documents in the *DOCNAM* field.
- *DOCNAM* - document name. The 1 - 12 character user assigned name of the document within the folder.
- *DESC* - document description. The description associated with the document.
- *LASTREV* - document last revision date. This is the date the document was either edited or details about the document (such as author, subject, or keywords) were changed. It is in the format YYYYMMDD where:

  YYYY = year
  MM = month
  DD = day

- *DOCTYPE* - document type. This is a keyword that describes the data stream content of the document. It may have the following values:
  - OTHER - unknown
  - FFTDCA - final-form text document
  - RFTDCA - revisable-form text document
  - PCFILE - personal computer data file
  - RFTS38 - IBM S/38 revisable document
  - HARDCOPY - reference to a printed document
  - RFT5520 - IBM 5520 revisable-form text document
  - EBCDIC - word processing EBCDIC text
  - WRDPRCINFF - word processing information file
  - IMAGE - image data subset document
  - TEXT3730 - IBM 3730 text data stream
  - DOCDESCDOC - document library descriptor document
  - DATAST3732 - IBM 3732 display document data stream
  - DOCUNITCTL - DIA defined document unit control
  - PRT1403 - IBM 1403 printer compatible data stream
  - AUDIO - digitized ADS audio (voice)

- RFTAS400 - AS/400 revisable-form document
- FFTAS400 - AS/400 final-form document
- RFTDW4 - IBM DW4 revisable-form document
- ASCIIDATA - ASCII data

File Document (FILDOC) File Considerations

The File Document command will accept a data base file as input. The file may be a source file or data file with no office considerations. In this case, the data is read and filed into the document library.

The file could also be a data file that is formatted to be the same as the outfile that the Retrieve Document (RTVDOC) command creates or the file may be a data file that is formatted to be the same as the outfile that the Receive Distribution (RCVDST) command creates. When the file is in this form, the File Document command will recognize that and properly process the file to format the data and the interchange document profile (IDP).

Thus it is possible to issue a Retrieve Document command to copy a document to a file or receive a distribution and later file that document back in the document library using the File Document command. It is also possible to write your own application to format a data file in the same manner as the Retrieve Document command or the Receive Distribution command does and use that file as input to the File Document command.

Print Document (PRTDOC) Outfile Considerations

The Print Document command can be used to create a data base file that contains descriptive information about the document (such things as authors, subject, or action due date).

The model outfile supplied with the system is named QAOPOUFL in library QSYS.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QAOPOUFL in library QSYS
A*  File Type:  Physical
A*  Record Format Name:  RECORD
A*
A*
A          R RECORD
A            FIELD              256A
A*
A* End of specifications
```

The meaning of each field in the record is:

- *FIELD* - data field. See the following tables for how the data field is defined for each record written to the file.

File Format for PRTFILE Files

For each document copied to an outfile, one header record is created as the first record in that file. All records in the outfile have a record length of 256 bytes.

Header Record

The header record contains the following fields:

Beginning Column	Field Length	Character (C) or Binary (B)	Contents or Description
1	1	B	X'20'
2	2	C	PF
4	253	C	Unused (unused positions are set to blanks)

Following the header record are the document records (record identifiers 10 through 99). There will be a start document record as the first record for each document printed to the file.

Start Document Record

The start document record contains the following fields:

Beginning Column	Field Length	Character (C) or Binary (B)	Contents or Description
1	2	C	Record identifier (10)
3	4	C	Form (PF = print file)
7	12	C	Document name
19	8	C	Folder name
27	62	C	Subdirectory
89	12	C	12-character folder name
101	156	C	Unused (unused positions are set to blanks)

Document Description Records

Following the document record are document description records. The first document description record contains the following fields:

Beginning Column	Field Length	Character (C) or Binary (B)	Contents or Description
1	2	C	Record identifier (11)
3	60	C	Subject
63	35	C	Author
98	35	C	Receiver
133	1	C	Unused (unused positions are set to blanks)
134	16	C	Document class
150	8	C	Date complete
158	8	C	Date last revised
166	8	C	Creation date
174	8	C	Action due date
182	8	C	Expiration date
190	10	C	Project
200	10	C	Reference
210	16	C	Keyword
226	1	C	Status
227	2	B	Internal status ('8000'X = print as labels, '4000'X = object (graphics), '0000'X = other)
229	6	C	Time last revised
235	22	C	Unused (unused positions are set to blanks)

The second document description record contains the following fields:

Beginning Column	Field Length	Character (C) or Binary (B)	Contents or Description
1	2	C	Record identifier (12)
3	35	C	Document description
38	60	C	Capitalized subject
98	13	C	Original system
111	44	C	Original name
155	4	C	Profile code page
159	4	C	Document code page
163	1	C	Unused (unused positions are set to blanks)
164	1	C	Unused (unused positions are set to blanks)
165	2	B	Document type
167	17	C	Unused (unused positions are set to blanks)
184	184	C	Unused (unused positions are set to blanks)
192	192	C	Unused (unused positions are set to blanks)
200	6	C	Unused (unused positions are set to blanks)
206	1	C	Unused (unused positions are set to blanks)
207	2	B	Type available Bit 16 = AS/400 unresolved Bit 15 = AS/400 resolved Bit 14 = L2-DCA (FFT) Bit 13 = L3-DCA (RFT) Bit 12 = PC data Bit 11 = DW/4 internal
209	1	C	PC mark for archive (1 = yes, 2 = no)
210	1	C	PC read only (1 = yes, 2 = no)
211	1	C	Unused (unused positions are set to blanks)
212	1	C	Unused (unused positions are set to blanks)
213	44	C	Unused (unused positions are set to blanks)

The third document description record contains the following fields:

Beginning Column	Field Length	Character (C) or Binary (B)	Contents or Description
1	2	C	Record identifier (13)
3	16	C	Document owner
19	8	C	Date filed
27	4	C	Time filed
31	8	C	User document date
39	60	C	Reference
99	20	C	Status
119	60	C	Cabinet reference
179	1	C	Archive mark
180	1	C	Work in progress
181	72	C	Unused (set to blanks)

The fourth document description record contains the following fields:

Beginning Column	Field Length	Character (C) or Binary (B)	Contents or Description
1	2	C	Record identifier (14)
3	240	C	60-character subject may occur 4 times
243	14	C	Unused (set to blanks)

The fifth document description record contains the following fields:

Beginning Column	Field Length	Character (C) or Binary (B)	Contents or Description
1	2	C	Record identifier (15)
3	240	C	20-character author may occur 12 times
243	14	C	Unused (set to blanks)

The sixth document description record contains the following fields:

Beginning Column	Field Length	Character (C) or Binary (B)	Contents or Description
1	2	C	Record identifier (16)
3	245	C	35-character keywords may occur 7 times
248	9	C	Unused (set to blanks)

The seventh document description record contains the following fields:

Beginning Column	Field Length	Character (C) or Binary (B)	Contents or Description
1	2	C	Record identifier (17)
3	240	C	60-character recipient may occur 4 times
243	14	C	Unused (set to blanks)

Query Document Library (QRYDOCLIB) Outfile Considerations

The Query Document Library command can be used to create a data base file that contains information about documents in the document library that have satisfied some query criteria.

The model outfile supplied with the system is named QAOSIQDL in library QSYS.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QAOSIQDL in library QSYS
A*  File Type:  Physical
A*  Record Format Name:  OSQDL
A*
A*
A          R OSQDL
A            QDLSNO         6S        TEXT('Sequence number')
A            QDLDID        24A        TEXT('Document +
A                                     identifier')
A            QDLDNM        12A        TEXT('Document name')
A            QDLFLR        63A        TEXT('Folder name')
A            QDLONM        10A        TEXT('Object name')
A            QDLUID         8A        TEXT('Owner user ID')
A            QDLADR         8A        TEXT('Owner user +
A                                     address')
A            QDLDSZ         8S        TEXT('Document size')
A            QDLDTP         5S        TEXT('Document type')
A            QDLSYC        13A        TEXT('System code')
A            QDLACF         1A        TEXT('Access flag')
A            QDLPER         1A        TEXT('Personal flag')
A            QDLWIP         1A        TEXT('Work in process flag')
A            QDLAHV         1A        TEXT('Mark for offline flag')
A            QDLCKO         1A        TEXT('Checkout flag')
A            QDLCRS         3S        TEXT('Data character set')
A            QDLCPG         3S        TEXT('Data code page')
A            QDLDL          3S        TEXT('Data length')
A            QDLRC          3S        TEXT('Record code')
A            QDLDTA       500A        TEXT('Data')
A*
A* End of specifications
```

The meaning of each field in the record is:

- *QDLSNO* - sequence number. A zoned numeric field that starts at 1 for the first record of document descriptor data and is incremented by 1 for the next record of a description.

- *QDLDID* - document identifier. This is the name assigned to the document by the system when it was filed. It consists of the system name concatenated with a time stamp. No two documents have the same document identifier. Each record for a document has the same document identifier. The Document Interchange Architecture refers to the document identifier as library assigned document name (LADN). It is in the format YYYYMMDDHHMNSSHSSNSNSNSNSNSNSNSN where:

  YYYY = year
  MM = month
  DD = day
  HH = hour
  MN = minute
  SS = second
  HS = hundredths of second
  SNSNSNSNSNSNSNSN = system name

- *QDLDNM* - document name. This is the unique document name specified by the user when the document was filed into the document library.
- *QDLFLR* - folder name. This is the folder path name to the folder containing the document. If the document is not in a folder, this field contains blanks.
- *QDLONM* - object name. This is the unique object name created by the system and used on other system commands (for example, DSPOBJD, DMPSYSOBJ).
- *QDLUID* - owner user ID. The user ID of the owner of the document. This data is put into code from code page 256/character set 930.
- *QDLADR* - owner address. The address of the owner of the document. This data is put into code from code page 256/character set 930.
- *QDLDSZ* - document size. This is the size in bytes of the document data.
- *QDLDTP* - document type. The document data is defined by this identifier.

DOCUMENT TYPE VALUE	DOCUMENT TYPE MEANING
2	Final-form text document (FFTDCA)
3	5520 revisable-form text document
4	Word-processing EBCDIC
5	Word-processing information file (WPIF)
6	Image-data subset document
7	3730 text data stream
8	DIA document library document descriptor document
9	3732 display document data stream
10	DIA defined document unit content
11	Revisable-form text document (RFTDCA)
12	1403 printer compatible data stream with variable length, unblocked records
13	Digitized ADS audio
14	IBM PC data file
15	Hard copy of document
223	Data file

  Also, refer to *Document Interchange Architecture: Interchange Document Profile Reference*, SC23-0764, for more details.
- *QDLSYC* - document system code. The document data is defined by this identifier. See the File Document (FILDOC) command for further information. This data is put into code from code page 256/character set 337.

- *QDLACF* - access flag. A 1-character flag to indicate if the document is a private or public document.
  - 0 = public
  - 1 = private
  - 9 = neither
- *QDLPER* - personal flag. A 1-character flag to indicate if the document is personal (not available to users working on behalf of the document owner).
  - 0 = not personal
  - 1 = personal
- *QDLWIP* - work in process flag. A 1-character flag to indicate a revisable (work in process document) or a final (non-replaceable) document.
  - 0 = final
  - 1 = revisable
- *QDLAHV* - save flag. A 1-character flag to indicate if a document has been marked for saving to offline storage.
  - 0 = not marked for save
  - 1 = marked for save only
  - 2 = marked for save and delete document object
  - 3 = marked for save and delete all data
- *QDLCKO* - checked out flag. A 1-character flag to indicate if a revisable document has been checked out for update.
  - 0 = not checked out
  - 1 = checked out for update.
- *QDLCRS* - data field character set. The data in the data field (*QDLDTA*) is put into code in this character set.
- *QDLCPG* - data field code page. The data in the data field (*QDLDTA*) is put into code in this code page.
- *QDLDL* - length of data. This number defines the length of the data field (*QDLDTA*). It encompasses only the data field. For some records, the length is not really necessary because the following data is a fixed length. For variable length data it is necessary. It is correctly set for all records.
- *QDLRC* - record code. A unique zoned number is defined for each record type. The record code must be used to determine the definition of the data to follow.
- *QDLDTA* - data field. This field takes on many meanings and is further described in the following:

Record Code	Data Field	Description
105	QDLDTA	Document description. This is a 1- to 44-character document name assigned by the user. The system applies no meaning to this description. QDLCRS and QDLCPG define the character set and code page of this data.
110	QDLDTA	Creation date and time. This is the date and time the document was created. This date and time are contained in 12 characters in the format YYYYMMDDHHMN where: YYYY = year MM = month DD = day HH = hour MN = minute
115	QDLDTA	Expiration date. This is the date when the document is no longer needed. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
120	QDLDTA	Document date. This is any unique date the user wishes to associate with the document. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
125	QDLDTA	File date and time. This is the date and time the document was filed in the system's document library. This date and time were automatically created by the system when the document was filed. This date and time are contained in 16 characters in the format YYYYMMDDHHMNSSHS where: YYYY = year MM = month DD = day HH = hour MN = minute SS = second HS = hundredths of second
130	QDLDTA	Date and time last changed. This is the date and time the document was last changed. This date and time are contained in 12 characters in the format YYYYMMDDHHMN where: YYYY = year MM = month DD = day HH = hour MN = minute

Record Code	Data Field	Description
135	QDLDTA	Action due date. This is the date when the action specified in the document is due. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
140	QDLDTA	Completion date. This is the date when the action specified in the document is complete. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
145	QDLDTA	Author name. This is the name of the author of the document (1 to 20 characters). QDLCRS and QDLCPG define the character set and code page of this data.
150	QDLDTA	Copy name. This is the name of one entry in the copy list (1 to 60 characters). QDLCRS and QDLCPG define the character set and code page of this data.
155	QDLDTA	Document class. This is the document class (1 to 16 characters). QDLCRS and QDLCPG define the character set and code page of this data.
160	QDLDTA	File cabinet reference. This refers to offline storage where a printed document resides (1 to 60 characters). QDLCRS and QDLCPG define the character set and code page of this data.
165	QDLDTA	Subject. This is the subject of the document (1 to 60 characters). QDLCRS and QDLCPG define the character set and code page of this data.
170	QDLDTA	Keyword. This is a keyword associated with the document (1 to 35 characters). QDLCRS and QDLCPG define the character set and code page of this data.
175	QDLDTA	Reference. This is the reference associated with the document (1 to 60 characters). QDLCRS and QDLCPG define the character set and code page of this data.
180	QDLDTA	Status. This is the user-defined work progress descriptor associated with the document (1 to 20 characters). QDLCRS and QDLCPG define the character set and code page of this data.

Record Code	Data Field	Description
185	QDLDTA	Project. This is the user-defined work identifier associated with the document (1 to 10 characters). QDLCRS and QDLCPG define the character set and code page of this data.
500	QDLDTA	IDP data. This is the interchange document profile (IDP) that is kept with the document. It conforms to the data stream definition of the Document Interchange Architecture. Refer to *Document Interchange Architecture: Interchange Document Profile Reference*, SC23-0764, for more information. This data is optionally provided for the user that needs to access the IDP for fields other than those provided by separate record codes. It is the user's responsibility to parse this data and extract the desired information. This field will be 500 bytes of the IDP per record. The last record with this record code may have less than 500 bytes. The data length is defined by QDLDL.

Query Distribution (QRYDST) Outfile Considerations

The Query Distribution command can be used to create a data base file that contains information about incoming distributions currently queued by the system waiting for the recipient to obtain them. The Query Distribution command can also be used to create a data base file that contains information about distributions the user has sent.

Incoming Distribution Information

The model outfile supplied with the system is named QAOSILIN in library QSYS and is used for incoming distribution information.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QAOSILIN in library QSYS
A*  File Type:  Physical
A*  Record Format Name:  OSLIN
A*
A*
A          R OSLIN
A            LINSNO         6S         TEXT('Sequence number')
A            LINDID        20A         TEXT('Distribution +
A                                       identifier')
A            LINRUI         8A         TEXT('Recipient identifier')
A            LINRUA         8A         TEXT('Recipient address')
A            LINSUI         8A         TEXT('Sender identifier')
A            LINSUA         8A         TEXT('Sender address')
A            LINSDT         8A         TEXT('Send date')
A            LINSTM         6A         TEXT('Send time')
A            LINPER         1A         TEXT('Personal flag')
A            LINPTY         1A         TEXT('Priority flag')
A            LINCOD         1A         TEXT('Confirmation of +
A                                       delivery flag')
A            LINDTP         5S         TEXT('Document type')
A            LINSYC        13A         TEXT('System code')
A            LINCID         3S         TEXT('Distribution +
A                                       description character set')
A            LINCPG         3S         TEXT('Distribution +
A                                       description code page')
A            LINDSD        44A         TEXT('Distribution +
A                                       description')
A*
A* End of specifications
```

The meaning of each field in the record is:

- *LINSNO* - sequence number. A zoned numeric field that starts at 1 for the first record of a distribution and is incremented by 1 for the next record of a distribution. Note that this number continues to be incremented. It does not start over when the records of another distribution are written to the file.
- *LINDID* - distribution identifier. The unique identifier assigned to a distribution. It is composed of the user's address (8 characters), the user ID (8 characters), and a 4-byte zoned sequence number with leading zeros. For example: NEWYORK JACKSON 0023. Each record for a distribution has the same distribution ID. This data is put into code from code page 256/character set 930.
- *LINRUI* - recipient user ID. This is the user ID of the person who receives the distribution. This data is put into code from code page 256/character set 930.
- *LINRUA* - recipient user address. This is the address of the person who receives the distribution. This data is put into code from code page 256/character set 930.
- *LINSUI* - sender user ID. This is the user ID of the person who sent the distribution. This data is put into code from code page 256/character set 930.
- *LINSUA* - sender address. This is the address of the person who sent the distribution. This data is put into code from code page 256/character set 930.

- *LINSDT* - date distribution sent. This is the date that the distribution was sent by the sender. If the sender is on another office system node, this is the date from that system. It is in the format YYYYMMDD where:

  YYYY = year
  MM = month
  DD = day

- *LINSTM* - time distribution sent. This is the time that the distribution was sent by the sender. If the sender is on another office system node, this is the time from that system. It is in the format HHMNSS where:

  HH = hour
  MN = minute
  SS = second

- *LINPER* - personal distribution. A value of 1 indicates this is a personal distribution; 0 indicates it is not personal.
- *LINPTY* - distribution priority. A value of 1 indicates this was sent with high priority; 0 indicates it was sent with normal priority.
- *LINCOD* - confirmation of delivery. A value of 1 indicates that the sender asked for confirmation of delivery; 0 indicates that confirmation of delivery was not requested.
- *LINDTP* - document type. The document data is defined by this identifier.

DOCUMENT TYPE VALUE	DOCUMENT TYPE MEANING
2	Final-form text document (FFTDCA)
3	5520 revisable-form text document
4	Word-processing EBCDIC
5	Word-processing information file (WPIF)
6	Image-data subset document
7	3730 text data stream
8	DIA document library document descriptor document
9	3732 display document data stream
10	DIA defined document unit content
11	Revisable-form text document (RFTDCA)
12	1403 printer compatible data stream with variable length, unblocked records
13	Digitized ADS audio
14	IBM PC data file
15	Hard copy of document
223	Data file

  Also, refer to *Document Interchange Architecture: Interchange Document Profile Reference*, SC23-0764, for more details. If this was a message-only distribution, this field contains zeros.
- *LINSYC* - document system code. The document data is defined by this identifier. See the Send Distribution (SNDDST) command for further information. If this was a message-only distribution, this field contains blanks. This data is put into code from code page 256/character set 337.
- *LINCID* - character set. The distribution description is put into code in this character set. This will be all zeros if no distribution description was supplied by the sender.
- *LINCPG* - code page. The distribution description is put into code in this code page. This will be all zeros if no distribution description was supplied by the sender.

- *LINDSD* - distribution description. This is 44 characters of text supplied by the sender of the distribution. This will be all blanks if no description was supplied. *LINCID* and *LINCPG* define the character set and code page of this data.

Outgoing Distribution Information

The model outfile supplied with the system is named QAOSILOT in library QSYS and is used for outgoing distribution information. One record is written to the file for each recipient on the distribution.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QAOSILOT in library QSYS
A*  File Type:  Physical
A*  Record Format Names:  OSLOUT
A*
A*
A          R OSLOUT
A            OUTSNO         6S         TEXT('Sequence number')
A            OUTDID        20A         TEXT('Distribution +
A                                       identifier')
A            OUTSUI         8A         TEXT('Sender identifier')
A            OUTSUA         8A         TEXT('Sender address')
A            OUTSDT         8A         TEXT('Send date')
A            OUTSTM         6A         TEXT('Send time')
A            OUTPER         1A         TEXT('Personal flag')
A            OUTPTY         1A         TEXT('Priority flag')
A            OUTCOD         1A         TEXT('Confirmation of +
A                                       delivery flag')
A            OUTCID         3S         TEXT('Distribution +
A                                       description character set')
A            OUTCPG         3S         TEXT('Distribution +
A                                       description code page')
A            OUTDSD        44A         TEXT('Distribution +
A                                       description')
A            OUTRUI         8A         TEXT('Recipient identifier')
A            OUTRUA         8A         TEXT('Recipient address')
A            OUTSTS         4S         TEXT('Status code')
A            OUTTYP         4S         TEXT('Status type')
A            OUTDAT         8A         TEXT('Status date')
A            OUTTIM         6A         TEXT('Status time')
A*
A* End of specifications
```

The meaning of each field in the record is:

- *OUTSNO* - sequence number. A zoned numeric field that starts at 1 for the first record of a distribution and is incremented by 1 for the next record of a distribution. Note that this number continues to be incremented. It does not start over when the records of another distribution are written to the file.
- *OUTDID* - distribution identifier. The unique identifier assigned to a distribution. It is composed of the user's address (8 characters), the user ID (8 characters), and a 4-byte zoned sequence number with leading zeros. For example: NEWYORK JACKSON 0023. Each record for a distribution has the same distribution ID. This data is put into code from code page 256/character set 930.
- *OUTSUI* - sender's user ID. This is the user ID of the person who sent the distribution. This data is put into code from code page 256/character set 930.

- *OUTSUA* - sender's address. This is the address of the person who sent the distribution. This data is put into code from code page 256/character set 930.
- *OUTSDT* - date distribution sent. This is the date that the distribution was sent by the sender. It is in the format YYYYMMDD where:

  YYYY = year
  MM = month
  DD = day

- *OUTSTM* - time distribution sent. This is the time that the distribution was sent by the sender. It is in the format HHMNSS where:

  HH = hour
  MN = minute
  SS = second

- *OUTPER* - personal distribution. A value of 1 indicates this is a personal distribution; a value of 0 indicates it is not personal.
- *OUTPTY* - distribution priority. A value of 1 indicates this was sent with high priority; a value of 0 indicates it was sent with normal priority.
- *OUTCOD* - confirmation of delivery. A value of 1 indicates that the sender asked for confirmation of delivery. A value of 0 indicates that confirmation of delivery was not requested.
- *OUTCID* - character set. The distribution description is put into code in this character set. This will be all zeros if no distribution description was supplied by the sender.
- *OUTCPG* - code page. The distribution description is put into code in this code page. This will be all zeros if no distribution description was supplied by the sender.
- *OUTDSD* - distribution description. This is 44 characters of text supplied by the sender of the distribution. This will be all blanks if no description was supplied. *OUTCID* and *OUTCPG* define the character set and code page of this data.
- *OUTRUI* - recipient user ID. The user ID of a recipient or the first part of a distribution list ID. This data is put into code from code page 256/character set 930.
- *OUTRUA* - recipient address. The user address of a recipient or the second part of a distribution list ID. This data is put into code from code page 256/character set 930.
- *OUTTYP* - status type. The type of status that is set in the next field.
  - 0 - No status is set, distribution is in process
  - 1 - SNADS status
  - 512 - DIA status
- *OUTSTS* - status code. A code that describes the current status of the distribution for this recipient. The following values are defined:

Zoned Numeric Value	Description
0	Distribution is in process
1	SNADS routing error
2	SNADS detected invalid user id
3	SNADS hop count exceeded
4	SNADS detected syntax error
5	SNADS function not supported
6	SNADS permanent server error
7	SNADS unknown server name
8	SNADS invalid server parameters
9	SNADS invalid destination transaction program name
12	SNADS distribution purged
13	SNADS some user IDs lost
14	SNADS resource not available
15	SNADS system error
16	SNADS temporary server error
17	SNADS system I/O error
18	SNADS Acknowledge Interchange Unit error
19	SNADS distribution object size inconsistency error
257	DIA distribution was delivered
513	DIA a system error has occurred
769	DIA invalid local recipient
1025	DIA invalid document
1281	DIA recipient canceled the distribution
1282	DIA distribution was removed prior to delivery to the recipient
1537	DIA maximum list exceeded
1793	DIA some recipients canceled/purged and some delivered
1794	DIA some recipients delivered and some invalid
1795	DIA some recipients canceled/removed and some invalid
1796	DIA some recipients canceled/removed, some delivered, & some invalid
1797	DIA some recipients canceled and some removed

Note: 17xx status codes apply when the recipient user ID was really a distribution list on a remote system. If all recipients on the remote distribution list have the same status, the common status code will be returned. For example, if all the recipients on the remote distribution list cancel the distribution, the status code will be 1281. But if the recipients on the remote distribution list have different status, one of the 17xx status codes will be returned for the status of the list.

- *OUTDAT* - date status set. If the distribution has been delivered or error status has been returned, this is the date that this occurred. If the recipient is on another office system node, this is the date from that system. If no status has been returned yet, this field will contain zeros. It is in the format YYYYMMDD where:

  YYYY = year
  MM = month
  DD = day

- *OUTTIM* - time status set. If the distribution has been delivered or error status has been returned, this is the time that this occurred. If the recipient is on another office system node, this is the time from that system. If no status has been returned yet, this field will contain zeros. It is in the format HHMNSS where:

  HH = hour
  MN = minute
  SS = second

Receive Distribution (RCVDST) Outfile Considerations

The Receive Distribution command can be used to create a data base file that contains a distribution sent to an office user. The distribution information as well as the distribution itself (for example, document or PC file) is written to the file.

The model outfile supplied with the system is named QAOSIRCV in library QSYS.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QAOSIRCV in library QSYS
A*  File Type:  Physical
A*  Record Format Name:  OSRCVD
A*
A*
A          R OSRCVD
A            RCVSNO         6S         TEXT('Sequence number')
A            RCVDID        20A         TEXT('Distribution +
A                                       identifier')
A            RCVRUI         8A         TEXT('Recipient +
A                                       identifier')
A            RCVRUA         8A         TEXT('Recipient address')
A            RCVSUI         8A         TEXT('Sender identifier')
A            RCVSUA         8A         TEXT('Sender address')
A            RCVSDT         8A         TEXT('Send date')
A            RCVSTM         6A         TEXT('Send time')
A            RCVPER         1A         TEXT('Personal flag')
A            RCVPTY         1A         TEXT('Priority flag')
A            RCVCOD         1A         TEXT('Confirmation of +
A                                       delivery flag')
A            RCVDSZ         8S         TEXT('Document size')
A            RCVDTP         5S         TEXT('Document type')
A            RCVSYC        13A         TEXT('System code')
A            RCVCRS         3S         TEXT('Data character set')
A            RCVCPG         3S         TEXT('Data code page')
A            RCVDL          3S         TEXT('Data length')
A            RCVRC          3S         TEXT('Record code')
A            RCVDTA       500A         TEXT('Data')
A*
A* End of specifications
```

The meaning of each field in the record is:

- *RCVSNO* - sequence number. A zoned numeric field that starts at 1 for the first record of a distribution and is incremented by 1 for the next record of a distribution.
- *RCVDID* - distribution identifier. The unique identifier assigned to a distribution. It is composed of the user's address (8 characters), the user ID (8 characters), and a 4 byte zoned sequence number with leading zeros. For example: NEWYORK JACKSON 0023. Each record for a distribution has the same distribution ID. This data is put into code from code page 256/character set 930.
- *RCVRUI* - recipient user ID. This is the user ID of the person who received the distribution. This data is put into code from code page 256/character set 930.
- *RCVRUA* - recipient address. This is the address of the person who received the distribution. This data is put into code from code page 256/character set 930.
- *RCVSUI* - sender's user ID. This is the user ID of the person who sent the distribution. This data is put into code from code page 256/character set 930.
- *RCVSUA* - sender's address. This is the address of the person who sent the distribution. This data is put into code from code page 256/character set 930.
- *RCVSDT* - date distribution sent. This is the date that the distribution was sent by the sender. If the sender is on another office system node, this is the date from that system. It is in the format YYYYMMDD where:

  YYYY = year
  MM = month
  DD = day

- *RCVSTM* - time distribution sent. This is the time that the distribution was sent by the sender. If the sender is on another office system node, this is the time from that system. It is in the format HHMNSS where:

  HH = hour
  MN = minute
  SS = second

- *RCVPER* - personal distribution. A value of 1 indicates this is a personal distribution; 0 indicates it is not personal.
- *RCVPTY* - distribution priority. A value of 1 indicates this was sent with high priority; 0 indicates it was sent with normal priority.
- *RCVCOD* - confirmation of delivery. A value of 1 indicates that the sender asked for confirmation of delivery. A value of 0 indicates that confirmation of delivery was not requested.
- *RCVDSZ* - document size. This is the size in bytes of the document data.
- *RCVDTP* - document type. The document data is defined by this identifier.

DOCUMENT TYPE VALUE	DOCUMENT TYPE MEAINING
2	Final-form text document (FFTDCA)
3	5520 revisable-form text document
4	Word-processing EBCDIC
5	Word-processing information file (WPIF)
6	Image-data subset document
7	3730 text data stream
8	DIA document library document descriptor document
9	3732 display document data stream
10	DIA defined document unit content
11	Revisable-form text document (RFTDCA)
12	1403 printer compatible data stream with variable length, unblocked records
13	Digitized ADS audio
14	IBM PC data file
15	Printed copy of document
223	Data file

Also, refer to *Document Interchange Architecture: Interchange Document Profile Reference*, SC23-0764, for more details. If this was a message-only distribution, this field contains zeros.

- *RCVSYC* - document system code. The document data is defined by this identifier. This data is put into code from code page 256/character set 930. See the Send Distribution (SNDDST) command for further information.
- *RCVCRS* - character set. The data field (*RCVDTA*) is put into code in this character set. This field is only applicable to some record codes, see record codes below for more information.
- *RCVCPG* - code page. The data field (*RCVDTA*) is put into code in this code page. This field is only applicable to some record codes, see record codes below for more information.
- *RCVDL* - data length. This is the length of data in the data field (*RCVDTA*) for this record.
- *RCVRC* - record code. This is the record code for this record. The content of the following field is determined by this code.
- *RCVDTA* - data field. The contents of this field are described below.

Record Code	Data Field	Description
010	RCVDTA	Distribution description. This is 44 characters of text supplied by the sender of the distribution. RCVCRS and RCVCPG define the character set and code page of this data.
020	RCVDTA	Message. This is a 1- to 256-character message associated with the distribution. RCVCRS and RCVCPG define the character set and code page of this data.

Record Code	Data Field	Description
105	RCVDTA	Document description. This is a 1- to 44-character name assigned to the document. The system applies no meaning to this description. RCVCRS and RCVCPG define the character set and code page of this data.
110	RCVDTA	Creation date and time. The date and time the document was created. This date and time are contained in 12 characters in the format YYYYMMDDHHMN where: YYYY = year MM = month DD = day HH = hour MN = minute
115	RCVDTA	Expiration date. This is the date when the document is no longer needed. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
120	RCVDTA	Document date. This is any unique date the user wishes to associate with the document. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
125	RCVDTA	File date and time. This is the date and time the document was filed in the system's document library. This date and time were automatically created by the system when the document was filed. This date and time are contained in 16 characters in the format YYYYMMDDHHMNSSHS where: YYYY = year MM = month DD = day HH = hour MN = minute SS = second HS = hundredths of second
130	RCVDTA	Date and time last changed. This is the date and time the document was last changed. This date and time are contained in 12 characters in the format YYYYMMDDHHMN where: YYYY = year MM = month DD = day HH = hour MN = minute

Record Code	Data Field	Description
135	RCVDTA	Action due date. This is the date when the action specified in the document is due. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
140	RCVDTA	Completion date. This is the date when the action specified in the document is complete. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
145	RCVDTA	Author name. This is the name of the author of the document (1 to 20 characters). RCVCRS and RCVCPG define the character set and code page of this data.
150	RCVDTA	Copy name. This is the name of one entry in the copy list (1 to 60 characters). RCVCRS and RCVCPG define the character set and code page of this data.
155	RCVDTA	Document class. This is the document class (1 to 16 characters). RCVCRS and RCVCPG define the character set and code page of this data.
160	RCVDTA	File cabinet reference. This refers to offline storage where a printed document resides (1 to 60 characters). RCVCRS and RCVCPG define the character set and code page of this data.
165	RCVDTA	Subject. This is the subject of the document (1 to 60 characters). RCVCRS and RCVCPG define the character set and code page of this data.
170	RCVDTA	Keyword. This is a keyword associated with the document (1 to 35 characters). RCVCRS and RCVCPG define the character set and code page of this data.
175	RCVDTA	Reference. This is the reference associated with the document (1 to 60 characters). RCVCRS and RCVCPG define the character set and code page of this data.
180	RCVDTA	Status. This is the user-defined work progress descriptor associated with the document (1 to 20 characters). RCVCRS and RCVCPG define the character set and code page of this data.

Record Code	Data Field	Description
185	RCVDTA	Project. This is the user-defined work identifier associated with the document (1 to 10 characters). RCVCRS and RCVCPG define the character set and code page of this data.
500	RCVDTA	IDP data. This is the interchange document profile (IDP) that was distributed with the document. It conforms to the data stream definition of the Document Interchange Architecture. Refer to *Document Interchange Architecture: Interchange Document Profile Reference*, SC23-0764, for details. This data is optionally provided for the user that needs to access the IDP for fields other than those provided by separate record codes. It is the user's responsibility to parse this data and extract the desired information. This field will be 500 bytes of the IDP per record. The last record with this record code may have less than 500 bytes. The data length is defined by RCVDL.
800	RCVDTA	Document data. This is the data that was distributed and is not defined by the CL command. The meaning of the data stream is defined by the document type and the system code. This field will contain 500 bytes of the document per record. The last record with this record code may have less than 500 bytes. The data length is defined by RCVDL. RCVCRS and RCVCPG define the character set and code page of this data. RFTDCA and FFTDCA data streams are described in *Document Content Architecture: Revisable-Form-Text Reference*, SC23-0758, and *Document Architecture: Final-Form-Text Reference*, SC23-0757.

Replace Document (RPLDOC) File Considerations

The Replace Document command will accept a data base file as input. The file can be a source file or data file with no office considerations. In this case, the data is read and replaces a document in the document library.

The file can also be a data file that is formatted to be the same as the outfile that the Retrieve Document (RTVDOC) command creates or the file may be a data file that is formatted to be the same as the outfile that the Receive Distribution (RCVDST) command creates. When the file is in this form, the Replace Document command will recognize that and properly process the file to format the data and the interchange document profile (IDP).

Thus it is possible to issue a Retrieve Document command to copy a document to a file and later file that document back in the document library using the Replace Document command or to receive a distribution and use the outfile of that command to replace a document in the document library. It is also possible to write your own application to format a data file in the same manner as the Retrieve Document command does and use that file as input to the Replace Document command.

Retrieve Document (RTVDOC) Outfile Considerations

The Retrieve Document command can be used to create a data base file that contains the document and document profile information that has been copied from the document library.

The model outfile supplied with the system is named QAOSIRTV in library QSYS.

The following data description specifications were used to create the model outfile:

```
A* Start of specifications
A*
A*  File Name:  QAOSIRTV in library QSYS
A*  File Type:  Physical
A*  Record Format Name:  OSRTVD
A*
A*
A          R OSRTVD
A            RTVSNO          6S        TEXT('Sequence number')
A            RTVDID         24A        TEXT('Document +
A                                      identifier')
A            RTVDNM         12A        TEXT('Document name')
A            RTVFLR         63A        TEXT('Folder name')
A            RTVONM         10A        TEXT('Object name')
A            RTVUID          8A        TEXT('Owner user ID')
A            RTVADR          8A        TEXT('Owner address')
A            RTVDSZ          8S        TEXT('Document size')
A            RTVDTP          5S        TEXT('Document type')
A            RTVSYC         13A        TEXT('System code')
A            RTVACF          1A        TEXT('Access flag')
A            RTVPER          1A        TEXT('Personal flag')
A            RTVWIP          1A        TEXT('Work in process +
A                                      flag')
A            RTVAHV          1A        TEXT('Mark for offline flag')
A            RTVCKO          1A        TEXT('Checkout flag')
A            RTVCRS          3S        TEXT('Data character set')
A            RTVCPG          3S        TEXT('Data code page')
A            RTVDL           3S        TEXT('Data length')
A            RTVRC           3S        TEXT('Record code')
A            RTVDTA        500A        TEXT('Data')
A*
A* End of specifications
```

- *RTVSNO* - Sequence number. A zoned numeric field that starts at 1 for the first record of document data and is incremented by 1 for the next record.

- *RTVDID* - document identifier. This is the name assigned to the document by the system when it was filed. It consists of the system name concatenated with a time stamp. No two documents have the same document identifier. Each record for a document has the same document identifier. The Document Interchange Architecture refers to the document identifier as library assigned document name (LADN). It is in the format YYYYMMDDHHMNSSHSSNSNSNSNSNSNSNSN where:

  YYYY = year
  MM = month
  DD = day
  HH = hour
  MN = minute
  SS = second
  HS = hundredths of second
  SNSNSNSNSNSNSNSN = system name

- *RTVDNM* - document name. This is the unique document name specified by the user when the document was filed into the DIA library.
- *RTVFLR* - folder name. This is the folder path name to the folder containing the document. If the document is not in a folder, this field contains blanks.
- *RTVONM* - object name. This is the object name created by the system and used on other system commands (for example, DSPOBJD, DMPSYSOBJ).
- *RTVUID* - owner user ID. The user ID of the owner of the document. This data is put into code from code page 256/character set 930.
- *RTVADR* - owner address. The address of the owner of the document. This data is put into code from code page 256/character set 930.
- *RTVDSZ* - document size. This is the size in bytes of the document data.
- *RTVDTP* - document type. The document data is defined by this identifier.

DOCUMENT TYPE VALUE	DOCUMENT TYPE MEAINING
2	Final-form text document (FFTDCA)
3	5520 revisable-form text document
4	Word-processing EBCDIC
5	Word-processing information file (WPIF)
6	Image-data subset document
7	3730 text data stream
8	DIA document library document descriptor document
9	3732 display document data stream
10	DIA defined document unit content
11	Revisable-form text document (RFTDCA)
12	1403 printer compatible data stream with variable length, unblocked records
13	Digitized ADS audio
14	IBM PC data file
15	Printed copy of document
223	Data file

  Also, refer to *Document Interchange Architecture: Interchange Document Profile Reference*, SC23-0764, for more details.
- *RTVSYC* - document system code. The document data is defined by this identifier. See the File Document (FILDOC) command for further information. This data is put into code from code page 256/character set 337.

- *RTVACF* - access flag. A 1-character flag to indicate if the document is a private or public document.
  - 0 = public
  - 1 = private
  - 9 = neither
- *RTVPER* - personal flag. A 1-character flag to indicate if the document is personal (not available to affinity users)
  - 0 = not personal
  - 1 = personal
- *RTVWIP* - work in process flag. A 1-character flag to indicate a revisable (work in process) or a final (non-replaceable) document.
  - 0 = final
  - 1 = revisable
- *RTVAHV* - save flag. A 1-character flag to indicate if a document has been marked for saving to offline storage.
  - 0 = not marked for save
  - 1 = marked for save only
  - 2 = marked for save and delete document object
  - 3 = marked for save and delete all data
- *RTVCKO* - checked out flag. A 1-character flag to indicate if a revisable document has been checked out for update.
  - 0 = not checked out
  - 1 = checked out for update.
- *RTVCRS* - data field character set. The data in the data field is put into code in this character set.
- *RTVCPG* - data field code page. The data in the data field is put into code in this code page.
- *RTVDL* - length of data. This number defines the length of the data field. It encompasses only the data field. For some records, the length is not really necessary because the following data is a fixed length. For variable length data it is necessary. It is correctly set for all records.
- *RTVRC* - record code. A unique zoned number is defined for each record type. The record code must be used to determine the definition of the data to follow.
- *RTVDTA* - data field. This field takes on many meanings and is further described in the following table.

Record Code	Data Field	Description
105	RTVDTA	Document description. This is a 1- to 44-character document name assigned by the user. The system applies no meaning to this description. RTVCRS and RTVCPG define the character set and code page of this data.
110	RTVDTA	Creation date and time. This is the date and time the document was created. This date and time are contained in 12 characters in the format YYYYMMDDHHMN where: YYYY = year MM = month DD = day HH = hour MN = minute
115	RTVDTA	Expiration date. This is the date when the document is no longer needed. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
120	RTVDTA	Document date. This is any unique date the user wishes to associate with the document. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
125	RTVDTA	File date and time. This is the date and time the document was filed in the system's document library. This date and time were automatically created by the system when the document was filed. This date and time are contained in 16 characters in the format YYYYMMDDHHMNSSHS where: YYYY = year MM = month DD = day HH = hour MN = minute SS = second HS = hundredths of second
130	RTVDTA	Date and time last changed. This is the date and time the document was last changed. This date and time are contained in 12 characters in the format YYYYMMDDHHMN where: YYYY = year MM = month DD = day HH = hour MN = minute

Record Code	Data Field	Description
135	RTVDTA	Action due date. This is the date when the action specified in the document is due. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
140	RTVDTA	Completion date. This is the date when the action specified in the document is complete. This date is contained in 8 characters in the format YYYYMMDD where: YYYY = year MM = month DD = day
145	RTVDTA	Author name. This is the name of the author of the document (1 to 20 characters). RTVCRS and RTVCPG define the character set and code page of this data.
150	RTVDTA	Copy name. This is the name of one entry in the copy list (1 to 60 characters). RTVCRS and RTVCPG define the character set and code page of this data.
155	RTVDTA	Document class. This is the document class (1 to 16 characters).
160	RTVDTA	File cabinet reference. This contains any reference to offline storage where a printed document resides (1 to 60 characters). RTVCRS and RTVCPG define the character set and code page of this data.
165	RTVDTA	Subject. This is the subject of the document (1 to 60 characters), the character set and code page apply.
170	RTVDTA	Keyword. This is a keyword associated with the document (1 to 35 characters). RTVCRS and RTVCPG define the character set and code page of this data.
175	RTVDTA	Reference. This is the reference associated with the document (1 to 60 characters). RTVCRS and RTVCPG define the character set and code page of this data.
180	RTVDTA	Status. This is the user-defined work progress descriptor associated with the document (1 to 20 characters). RTVCRS and RTVCPG define the character set and code page of this data.
185	RTVDTA	Project. This is the user-defined work identifier associated with the document (1 to 10 characters). RTVCRS and RTVCPG define the character set and code page of this data.

Record Code	Data Field	Description
400	RTVDTA	Access code. This is the access code and the description of the access code associated with the document. The access code and description are in the format `nnnndescription` where nnnn is a 4-digit access code (0001 to 2047) followed by the 1- to 50-character description.
500	RTVDTA	Interchange document profile (IDP) data. This is the interchange document profile (IDP) that is kept with the document. It conforms to the data stream definition of the Document Interchange Architecture. Refer to *Document Interchange Architecture: Interchange Document Profile Reference*, SC23-0764, for details. This data is optionally provided for the user that needs to access the IDP for fields other than those provided by separate record codes. It is the user's responsibility to parse this data and extract the desired information. This field will be 500 bytes of the IDP per record. The last record with this record code may have less than 500 bytes. The data length is defined by RTVDL.
800	RTVDTA	Document data. This is the data that was distributed and is not defined by the CL command. The meaning of the data stream is defined by the document type and the system code. This field will contain 500 bytes of the document per record. The last record with this record code may have less than 500 bytes. The data length is defined by RTVDL. RTVCRS and RTVCPG define the character set and code page of this data. FFTDCA and RFTDCA data streams are documented in *Document Content Architecture: Revisable Form Text Reference*, SC23-0758, and *Document Content Architecture: Final Form Text Reference*, SC23-0757.

Send Distribution (SNDDST) File Considerations

The Send Distribution command will accept a data base file as input. The file can be a source file or data file with no office considerations. In this case, the data is read and sent within the office network.

The file can also be a data file that is formatted to be the same as the outfile that the Receive Distribution (RCVDST) command creates or the file may be a data file that is formatted to be the same as the outfile that the Retrieve Document (RTVDOC) command creates. When the file is in this form, the Send Distribution command will recognize that and properly process the file to format the data and the interchange document profile (IDP).

Thus it is possible to issue a Receive Distribution command to put a distribution in a file and send that distribution to someone using the Send Distribution command. Or you may use the Retrieve Document command to copy a document from the document library into a data base file and then later send it to someone using the Send Distribution command. It is also possible to write your own application to format a data file in the same manner as the Receive Distribution command does and use that file as input to the Send Distribution command.

Glossary

access method. A method used to read a record from, or to write a record into a file. Access can be sequential (records are referred to one after another in the order in which they appear in the file), it can be random (the individual records can be referred to in any order), or it can be dynamic (records can be accessed sequentially or randomly, depending on the specific request).

access path. The order that records in a data base file are organized for processing by a program. See *arrival sequence access path* and *keyed sequence access path.*

access path journaling. A method of recording changes to an access path as changes are made to the data in the data base file so that the access path can be recovered automatically by the system.

active record. Any record format that is currently displayed or an active subfile record. In a data base file, a nondeleted record.

add authority. A data authority that allows the user to add entries to an object; for example, add job entries to a job queue or add records to a file. Contrast with *delete authority.* See also *read authority* and *update authority.*

after-image. The contents of the physical file record after the data was changed by a write or an update operation. Contrast with *before-image.*

all authority. An object authority that allows the user to perform all operations on the object except those limited to the owner or controlled by authorization list management authority. The user can control the object's existence, specify the security for the object, and change the object. Contrast with *exclude authority.*

allocate. To reserve (lock) a resource for use in performing a specific task. Contrast with *deallocate.*

alphameric. Pertaining to the letters, A-Z; numbers, 0-9; and special symbols, $, #, @, ., or _.

alternative collating sequence. A user-defined collating sequence that replaces the standard EBCDIC collating sequence. See *collating sequence.*

application. A particular business task, such as inventory control or accounts receivable.

application program. A program used to perform a particular data processing task such as inventory control or payroll.

arrival sequence access path. An access path to a data base file that is arranged according to the order in which records are stored in the physical file. See also *keyed sequence access path* and *access path.*

AS/400 Business Graphics Utility (BGU). The IBM licensed program that can be used to design, plot, display, and print graphics.

AS/400 Office. The IBM licensed program for sending and receiving mail, scheduling appointments on calendars, maintaining directories of names and addresses, working on behalf of groups of users, and getting documents.

AS/400 Query. The IBM licensed program used to select, format, and analyze information from data files to produce reports and other files.

ascending key. The values by which data is arranged from the lowest value to the highest value of the key field in accordance with the rules for comparing data items. Contrast with *descending key.*

ascending key sequence. The arrangement of data in order from the lowest value of the key field to the highest value of the key field. Contrast with *descending key sequence.*

ascending sequence. The arrangement of data in order from the lowest value to the highest value, according to the rules for comparing data. Contrast with *descending sequence.*

audit trail. Data in the form of a chronological path linking a sequence of events that allows the system to trace the transactions that have affected the contents of a record, such as a customer account or item record.

authority checking. A function of the system that looks for and verifies a user's authority to an object.

authorize. Permit or give authority to.

auxiliary storage. All addressable storage other than main storage.

BASIC (beginner's all-purpose symbolic instruction code). A programming language with a small list of commands and a simple syntax, primarily designed for numeric applications.

batch. Pertaining to a group of jobs to be run on a computer sequentially with the same program with little or no operator action. Contrast with *interactive.*

batch job. A predefined group of processing actions submitted to the system to be performed with little or

no interaction between the user and the system. Contrast with *interactive job*.

before-image. The contents of a record in a physical file before the data is changed by a write, an update, or a delete operation. Contrast with *after-image*.

BGU. See *AS/400 Business Graphics Utility (BGU)*.

binary floating-point number. The conceptual form of a numeric value that contains a significand and a signed exponent. The number's numeric value is the signed product of the number's significand and 2 raised to the power of the number's exponent.

both field. A field that can be used for either input data or output data.

built-in function. A predefined function, such as a commonly used arithmetic function or a function necessary to high-level language compilers (for example, a function for manipulating character strings or converting data). It is automatically called by a built-in function reference.

Business Graphics Utility (BGU). See *AS/400 Business Graphics Utility (BGU)*.

byte. A group of eight adjacent bits. In the EBCDIC coding system, one byte can represent a character. In the double-byte coding system, two bytes represent a character.

change authority. An object authority that allows a user to perform all operations on the object except those limited to the owner or controlled by object existence authority and object management authority. The user can add, change, and delete entries in an object, or read the contents of an entry in the object. Change authority combines object operational authority and all the data authorities.

character. Any letter, number, or other symbol in the data character set that is part of the organization, control, or representation of data.

character constant. The actual character value (a symbol, quantity, or constant) in a source program that is itself data, instead of reference to a field that contains the data. Contrast with *numeric constant*.

character field. An area that is reserved for information that can contain any of the characters in the data character set. Contrast with *numeric field*.

checksum protection. A function that protects data stored in the system auxiliary storage pool from being lost because of the failure of a single disk. When checksum protection is in effect and a disk failure occurs, the system automatically reconstructs the data when the system program is loaded after the device is repaired.

CL. See *control language (CL)*.

close. The function that ends the connection between a file and a program, and ends the processing. Contrast with *open*.

COBOL (common business-oriented language). A high-level programming language, based on English, that is used primarily for commercial data processing.

collating sequence. The order in which characters are arranged within the computer for sorting, combining, or comparing.

command. A statement used to request a function of the system. A command consists of the command name, which identifies the requested function, and parameters.

command line. The blank line on a display where commands, option numbers, or selections can be entered.

commit. To make all changes permanent that were made to one or more data base files since the last commit or rollback operation, and make the changed records available to other users.

commitment boundary. In a commitment controlled environment, any time there are no outstanding changes for a data base file existing within a job.

commitment control. A means of grouping file operations that allows the processing of a group of data base changes as a single unit through the Commit command or the removal of a group of data base changes as a single unit through the Rollback command.

compatibility. Ability to work in the system or ability to work with other devices or programs.

compile. To translate a program written in a high-level programming language into a machine language program.

compiled program. The set of machine language instructions that is the output from the compilation of a source program. The actual processing of data is done by the machine-language program.

compiler listing. A printout that is produced by compiling a program or creating a file and that optionally includes, for example, a line-by-line list of the high-level language source, a cross-reference list, diagnostic information; and for programs, the description of the externally described files. See also *source listing*.

completion message. A message that tells the operator when work is successfully ended.

composite key. A key for a file or record format that is composed of more than one key field.

compress. To replace repetitive characters in a file or folder with control characters so that the file or folder takes up less space when saved on diskette.

concatenate. (1) To link together. (2) To join two character strings.

concatenated field. Two or more fields that are combined to make one field in a logical file.

consecutive processing. A method of processing in which the records in the file are read, written to, or deleted in the order in which they exist in a file. See also *random processing* and *sequential processing*.

constant field. In an externally described display or printer file, an unnamed field that contains actual data that is passed to the display or printer but is unknown to the program passing it.

control language (CL). The set of all commands with which a user requests system functions.

control language program. A program that is created from source statements consisting entirely of control language commands.

control language variable. A program variable that is declared in a control language program and is available only to the CL program.

creation date. The system date when an object is created.

cross-reference listing. The part of the compiler listing that tells where files, fields, and indicators are defined, referred to, and changed in a program.

data authority. A specific authority to read, add, update, or delete data. See also *add authority*, *delete authority*, *read authority*, and *update authority*.

data base. The collection of all data files stored in the system.

data base file. An object that contains descriptions of how input data is to be presented to a program from internal storage and how output data is to be presented to internal storage from a program. See also *physical file* and *logical file*.

data definition. Information that describes the contents and characteristics of a field, record format, or file. A data definition can include such things as field names, lengths, and data types. See also *field definition*, and *file definition*.

data description specifications (DDS). A description of the user's data base or device files that is entered into the system in a fixed form. The description is then used to create files.

data dictionary. An object for storing field, record format, and file definitions.

data file utility (DFU). The part of the AS/400 Application Development Tools licensed program that is used to enter, maintain, and display records in a data base file.

data item. A unit of information to be processed.

data type. A characteristic used for defining data as numeric or character.

DBCS. See *double-byte character set (DBCS)*.

DBCS-either field. An input-capable field that may contain either double-byte data or alphameric data with selection made at entry time.

DBCS-only. An input-capable field that must contain only double-byte characters.

DBCS-open field. An input-capable or output field that may contain a mixture of alphameric and double-byte characters with shift-out and shift-in characters marking the transitions.

DDM. See *distributed data management (DDM)*.

DDS. See *data description specifications*.

deallocate. To release a resource that is assigned to a specific task. Contrast with *allocate*.

decimal position. (1) The location of the decimal point in a series of numbers. (2) Numbers to the right of the decimal point. For example, 4.009 has three decimal positions.

default record. A record that consists entirely of default values (numeric fields are filled with zeros, character fields are filled with blanks, or either data type, numeric or character, can be filled with a value specified by the user with the DFT keyword in DDS).

default value. A value supplied by the system that is used when no value is specified by the user, or the value specified by the user with the DFT keyword in DDS.

delayed maintenance. A method of logging changes to an access path for data base files and applying the changes the next time the file is opened instead of rebuilding the access path completely. Contrast with *rebuild maintenance* and *immediate maintenance*.

delete authority. A data authority that allows the user to remove entries from an object; for example, delete messages from a message queue or delete records from a file. Contrast with *add authority*. See also *read authority* and *update authority*.

descending key. The values by which data is arranged from the highest value to the lowest value of the key field, in accordance with the rules for comparing data items. Contrast with *ascending key*.

descending key sequence. The arrangement of data in order from the highest value of the key field to the lowest value of the key field. Contrast with *ascending key sequence*.

descending sequence. The arrangement of data in order from the highest value to the lowest value, according to the rules for comparing data. Contrast with *ascending sequence*.

DFU. See *data file utility (DFU)*.

diagnostic message. A message that contains information about errors or possible errors. This message is generally followed by an escape message.

digit. Any of the numerals from 0 through 9.

disk. A storage media made of one or more flat, circular sheets with magnetic surfaces on which information can be stored.

distributed data management (DDM). A function of the operating system that allows an application program or user on one system to use data files stored on remote systems. The systems must be connected by a communications network, and the remote systems must also be using DDM.

double precision. The specification that causes a floating-point value to be stored (internally) in the long format (two computer words). Contrast with *single precision*.

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, typing, displaying, and printing DBCS characters requires hardware and supporting programs that are DBCS-capable.

duplicate key value. The occurrence of the same value in a key field or in a composite key in more than one record in a file.

dynamic. Occurring at run time, or during processing.

dynamic select/omit. Selection and omission of logical file records performed during processing, instead of when the access path (if any) is maintained. Dynamic select/omit may also be used when no keyed access path exists.

EBCDIC. See *extended binary-coded decimal interchange code (EBCDIC)*.

EBCDIC character. Any one of the symbols included in the 8-bit EBCDIC set.

edit. To interactively add, change, delete, or rearrange the data; for example, to insert or remove characters, sentences, or paragraphs, or to insert or remove characters in dates or decimal numbers.

end-of-file delay. An interval during which the system holds a file open after the normal end of the file is reached until one or more records are updated or added to the end of the file. The length of the interval can be specified on the EOFDLY parameter.

escape message. A message that reports a condition that caused the program to end before the requested function was complete.

exclude authority. An object authority that prevents the user from using the object or its contents. Contrast with *all authority*.

exclusive allow-read lock state. The lock on an object that allows only one job to use the object, but allows other jobs to read the object. The predefined value for this lock state is *EXCLRD.

exclusive lock state. The lock on an object that allows only one job to use the object; no other job can use the object. The predefined value for this lock state is *EXCL.

extended binary-coded decimal interchange code (EBCDIC). A coded character set of 256 eight-bit characters.

externally described data. Data contained in a file for which the fields and the records are described outside of the using program (such as with DDS, IDDU, SQL/400), and used by a program when the file is processed. Contrast with *program-described data*.

externally described file. A file in which the record fields are described to the system when the file is created, and used by the program when the file is processed. Contrast with *program-described file*.

field. A group of related characters (such as name or amount) that are treated as a unit on a record.

field definition. (1) (IDDU)Information that describes the characteristics of data in a field. A field definition is contained in a data dictionary or in the file itself. (2) (DDS)Data description specifications that describe the characteristics of data in a field.

field level specifications. Specifications coded on the same line as a field name or on lines immediately following a field name. See also *file level specifications, record level specifications, help level specifications, join*

level specifications, *key field level specifications*, and *select/omit level specification*.

field reference file. A physical file that contains no data, only descriptions of fields.

file. A generic term for the object type that refers to a data base file, a device file, or a set of related records treated as a unit. The system-recognized identifier for the object type is *FILE.

file definition. Information that describes the contents and characteristics of a file. A file definition resides in a data dictionary.

file description. The data description specifications that describe the file and its contents.

file level specifications. Specifications coded on the lines before the first record format name. See also *field level specifications*, *key field level specifications*, *record level specifications*, *join level specifications*, *help level specifications*, and *select/omit level specifications*..

file name. The name used by a program to identify a file. See also *label*.

file organization. The permanent file arrangement established at the time that a file is created.

file overrides. Attributes specified at run time that change the attributes specified in the file description or in the program.

file reference function. A function of the system that lets the user track file use on the system.

fixed length. A specified length for a record or field that cannot be changed.

floating-point. A mathematical notation in which a quantity of a number is shown as one number multiplied by a power of the number of the base.

folder. A directory for documents. A folder is used to group related documents and to find documents by name. The system-recognized identifier for the object type is *FLR. Compare with *library*.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, files, or documents. (2) A group of related fields, such as a record, in a file.

format selector. A user-defined program (either a CL or a high-level language program) that determines where a record should be placed in the data base when an application program does not pass a record format name for a record being added to a logical file.

generic. Relating to, or characteristic of, a whole group or class.

group authority. Authority to use objects, resources, or functions from a group profile.

group job. One of up to sixteen interactive jobs that are associated in a group with the same work station device and user.

group profile. A user profile that provides the same authority to a group of users.

help level specifications. In a display file, specifications coded between the record and field level which define areas on the screen and associate help information with those areas. See also *file level specifications*, *field level specifications*, *join level specifications*, *key field level specifications*, *record level specifications*, and *select/omit level specifications*.

high-level language (HLL). A programming language, such as RPG, CL, BASIC, PL/I, Pascal, and COBOL, that translates computer programs in this language into several different machine codes.

history log. A summary of the system activities, such as system and job information, device status, system operator messages and a record of program temporary fix (PTF) activity on the system.

I/O. See *input/output*.

IDDU. See *interactive data definition utility (IDDU)*.

immediate maintenance. A method of maintaining keyed access paths for data base files. This method updates the access path whenever changes are made to the data base file associated with the access path. Contrast with *rebuild maintenance* and *delayed maintenance*.

implicit. Capable of being understood from something else, though unexpressed.

initial program load (IPL). The process that loads the system programs from the system auxiliary storage, checks the system hardware, and prepares the system for user operations.

inline data file. A file described by a Data command that is included as part of a job when the job is read from an input device. The file is erased when the job ends.

input field. A field specified in a display file or data base file that is used for data you supply. Contrast with *output field*.

input/output. Data provided to the computer or data resulting from computer processing.

inquiry message. A message that gives information and requests a reply.

interactive. Pertaining to the exchange of information between people and a computer. Contrast with *batch*.

interactive data definition utility (IDDU). A function of the operating system that can be used to externally define the characteristics of data and the contents of files.

interactive job. A job started for a person who signs on to a work station. Contrast with *batch job*.

IPL. See *initial program load (IPL)*.

job log. A record of requests submitted to the system by a job, the messages related to the requests, and the actions performed by the system on the job. The job log is maintained by the system program.

join. An operation that combines data from two or more files using specified fields.

join field. A comparison field that identifies records from two files to be combined into one record.

join file. A file with records made up of the fields from one or more physical files.

join level specifications. In a join logical file, specifications coded between the record and field level, which define how to join two physical files. See also *file level specifications*, *field level specifications*, *key field level specifications*, *help level specifications*, *record level specifications*, and *select/omit level specifications*.

join logical file. A logical file that combines (in one record format) fields from two or more physical files.

key. The value used to identify a record in a keyed sequence file.

key field. A field used to arrange the records of a particular type within a file member.

key field level specifications. In data description specifications (DDS), coded specifications on the lines following the last field specification. Key field level specifications are permitted only for physical files or logical files. See also *field level specifications*, *file level specifications*, *record level specifications*, *join level specifications*, *key field level specifications*, *help level specifications*, and *select/omit level specifications*.

keyed sequence. The order in which indexed records are read by the program.

keyed sequence access path. An access path to a data base file that is arranged according to the contents of key fields contained in the individual records. See also *arrival sequence access path* and *access path*.

keyword. A name that identifies a parameter. Keywords are used for parameter names in some CL commands.

keyword functions. The result of processing DDS keywords in a record format specified on an operation. See also *operation*.

label. The name of a file on a diskette or tape.

level checking. A function that compares the record format-level identifiers of a file to be opened with the file description that is part of a compiled program to determine if the record format for the file changed since the program was compiled.

library. An object on disk that serves as a directory to other objects. A library groups related objects, and allows the user to find objects by name. Compare with *folder*.

library list. A list that indicates which libraries are to be searched and the order in which they are to be searched. The system-recognized identifier is *LIBL.

link. To connect a data file on disk with a file definition in a data dictionary. Contrast with *unlink*.

lock state. A condition defined for an object that determines how it is locked, how it is used (read or write), and whether the object can be shared (used by more than one job).

logical file. A description of how data is to be presented to a program. This type of data base file contains no data, but it defines formats for one or more physical files. See also *join logical file*. Contrast with *physical file*.

logical file member. A named logical grouping of data records from one or more physical file members. See also *member*.

master file. A collection of permanent information, such as a file of customer addresses.

megabyte. A unit of measure for storage capacity; 1 megabyte = 1,048,576 bytes.

member. Different sets of data within one file.

numeric character. Any one of the digits 0 through 9.

numeric constant. The actual numeric value to be used in processing, instead of the name of a field containing the data. A numeric constant can contain any of the numeric digits 0 thorough 9, a sign (plus or minus), and a decimal point. Contrast with *character constant*.

numeric field. An area that is reserved for a particular unit of information and that can contain only the digits 0 through 9. Contrast with *character field*.

object. A named unit that consists of a set of characteristics that describe the object and, in some cases, data. An object is anything that exists in and occupies space in storage and on which operations can be performed, such as programs, files, libraries, and folders.

object authority. A specific authority that controls what a system user can do to an entire object. For example, object authority includes deleting, moving, or renaming an object. There are seven types of object authorities: object operational, object management, object existence, all, change, use, and exclude.

object existence authority. An object authority that allows the user to delete the object, free storage of the object, save and restore the object, transfer ownership of the object, and create an object that was named by an authority holder.

object management authority. An object authority that allows the user to specify the authority for the object, move or rename the object, and add members to data base files.

object operational authority. An object authority that allows the user to look at the description of an object and use the object as determined by the user's data authorities to the object. See also *all authority* and *use authority*.

object owner. A user who creates an object or to whom the ownership of an object was reassigned. The object owner has complete control over the object.

omit function. A system function that determines which records from a physical file are to be omitted from a logical file. Contrast with *select function*.

online information. Information, read on the display screen, that explains displays, messages, and programs.

open. The function that connects a file to a program for processing. Contrast with *close*.

open data path. The path that handles all input/output operations for the file.

operating system. A collection of system programs that control the overall operation of a computer system.

operation. The result of processing statements in a high-level language. See also *keyword functions*.

output field. A field specified in a display file or data base file that is reserved for the information processed by a program. Contrast with *input field*.

packed decimal format. Representation of a decimal value in which each byte within a field represents two numeric digits except the far right byte, which contains one digit in bits 0 through 3 and the sign in bits 4 through 7. For all other bytes, bits 0 through 3 represent one digit; bits 4 through 7 represent one digit. For example, the decimal value +123 is represented as 0001 0010 0011 1111. Contrast with *zoned decimal format*.

packed field. A field that contains data in the packed decimal format.

packed key. A key field in packed decimal format.

parameter. A value supplied to a command or program that either is used as input or controls the actions of the command or program.

physical file. A description of how data is to be presented to or received from a program and how data is actually stored in the data base. A physical file contains one record format and one or more members. Contrast with *logical file*.

physical file member. A named subset of the data records in a physical file. See also *member*.

PL/I. A programming language designed for use in a wide range of commercial and scientific computer applications.

private authority. The authority specifically given to a user for an object that overrides any other authorities, such as the authority of a user's group profile or an authorization list. Contrast with *public authority*.

processing. The action of performing operations and calculations on data.

production library. A library containing objects needed for normal processing. Contrast with *test library*.

program temporary fix (PTF). A temporary solution to, or bypass of, a defect in a current release of a licensed program.

program-described data. Data contained in a file for which the fields in the records are described in the program that processes the file. Contrast with *externally described data*.

program-described file. A file for which the fields in the records are described only in the program that processes the file. To the operating system, the record appears as a character string. Contrast with *externally described file*.

prompt. A list of values or a request for information provided by the system as a reminder of the type of information or action required.

public authority. The authority given to users who do not have any specific (private) authority to an object, who are not on the authorization list (if one is specified for the object), and whose group profile has no specific authority to the object. Contrast with *private authority*.

Query. The shortened name for the IBM AS/400 Query licensed program.

query. A request to select and copy from a file or files one or more records based on defined conditions. For example, a request for a list of all customers in a customer master file, whose balance is greater than $1,000.

queue. A list of messages, jobs, or files waiting to be read, processed, printed, or distributed in the order they appear in the list.

random by key. A processing method for files in which the value in the key field identifies the records to be processed.

random by relative record number. A processing method for files in which relative record numbers identify the records to be processed.

random processing. A method of processing in which records can be read from, written to, or deleted from a file order requested by the program that is using them. See also *consecutive processing* and *sequential processing*.

read authority. A data authority that allows the user to look at the contents of an entry in an object or to run a program. See also *add authority*, *delete authority*, and *update authority*.

read operation. An input operation that obtains a record from a file and passes it to a program.

rebuild maintenance. A method of maintaining keyed access paths for data base files. This method updates the access path only while the file is open, not when the file is closed; the access path is rebuilt when the file is opened. Contrast with *immediate maintenance* and *delayed maintenance*.

record. A collection of related data or words, treated as a unit; such as one name, address, and telephone number.

record level specifications. Specifications coded on the same line as a record format name or on lines immediately following a record format name (until the first field is specified). See also *field level specifications*, *file level specifications*, *key field level specifications*, *help level specifications*, *record level specifications*, *join level specifications*, and *select/omit level specifications*.

record selection. The process of selecting particular records from a file and including the information from the records, for example, in a report.

relational expression. A logical statement that describes the relationship (such as greater than or equal to) of two arithmetic expressions or data items.

relational operator. The reserved words or symbols used to express a relational condition or a relational expression.

relative file number. In a join logical file, a sequential number assigned to a physical file based on the position of that file on the JFILE keyword specification.

relative record number. A number that specifies the location of a record in relation to the beginning of a data base file, member, or subfile. For example, the first record in a data base file, member, or subfile has a relative record number of 1.

restore. To copy data from tape, diskette, or a save file to auxiliary storage. Contrast with *save*.

roll back. To remove changes that were made to files under commitment control since the last commitment boundary.

RPG. Report Program Generator. A programming language designed for writing application programs for business data processing requirements.

save. To copy specific objects or libraries by transferring them from main or auxiliary storage to magnetic media such as tape, diskettes, or a save file. Contrast with *restore*.

save file. A file allocated in auxiliary storage that can be used to store saved data on disk (without requiring diskettes or tapes), that can be used in I/O operations from a high-level language program, or can be used to receive objects sent through the network.

secure. Controlling who can use and to what extent an object can be used by controlling the authority given to the user.

select function. A system function that determines which records from a physical file are to be included in a logical file. Contrast with *omit function*.

select/omit field. A field in a logical file record format whose value is tested by the system to determine if records including that field are to be used. The test is a comparison with a constant, the contents of another field, a range of values, or a list of values; and the record is either selected or omitted as a result of the test. See also *dynamic select/omit*.

select/omit level specifications. Specifications coded on the lines following the last key-field specification. These specifications are permitted only in a logical file. See also *field level specifications*, *file level specifications*, *key field level specifications*, *record level specifications*, *help level specifications*, and *join level specifications*.

sequence number. (1) The number of a record that identifies the record within the source member. (2) A field in a journal entry that contains a number assigned by

the system. This number is initially 1 and is increased by 1 until the journal is changed or the sequence number is reset by the user.

sequential processing. A method of processing in which records are read, written to, or deleted in the order determined by the value of the key field. See also *consecutive processing* and *random processing*.

SEU. See *source entry utility (SEU)*.

shared access path. An access path used by more than one file member to get data common to both members.

shared file. A file whose open data path can be shared between two or more programs processing in the same routing step. See *open data path*.

shared record format. A record format that is used in more than one externally described file.

shared-for-read lock state. The lock state for a file in which the file can be shared with another program if the program does not request exclusive use of the file. The predefined value for this lock state is *SHRRD.

shared-for-update lock state. The lock state for a file in which the file can be shared either for update or for read operations with another program. The predefined value for this lock state is *SHRUPD.

shared-no-update lock state. The lock state for a file in which the file can be shared with another program if the program requests either a shared-no-update lock state or a shared-for-read lock state. The predefined value for this lock state is *SHRNUP.

significand. In binary floating-point format, the part of a number that contains the whole number and fraction.

single precision. The specification that causes the floating-point value to be stored (internally) in the short format. Contrast with *double precision*.

source entry utility (SEU). A function of the AS/400 Application Development Tools licensed program that is used to create and change source members.

source file. (1) A file of programming code that is not compiled into machine language. Contrast with *data file*. (2) A file created by the specification of FILETYPE(*SRC) on the Create command. A source file can contain source statements for such items as high-level language programs and data description specifications.

source listing. A portion of a compiler listing that contains source statements and, optionally, test results. See also *compiler listing*.

source member. A member of a data base source file that contains source statements such as RPG/400, COBOL, BASIC, PL/I, or DDS. See also *member*.

source program. A set of instructions that are written in a programming language and must be translated to machine language before the program can be run.

source statement. A statement written in symbols of a programming language. For example, RPG/400, COBOL, BASIC, PL/I, or DDS are source statements.

special authority. The types of authority a user can have to perform system functions, including all object authority, save system authority, job control authority, security administrator authority, spool control authority, and service authority. Contrast with *specific authority*.

specific authority. The types of authority a user can be given to use the system resources, including object authorities and data authorities. See also *object authority* and *data authority*. Contrast with *special authority*

spooled input file. See *inline data file*.

spooled output file. A file that causes output data to be held for later printing.

spooling. The system function that saves data in a disk file for later processing or printing.

statement. An instruction in a program.

Structured Query Language/400 (SQL/400). An IBM licensed program supporting the relational data base. Used to put information into a data base and to get and organize selected information from a data base.

substring. A part of a character string.

syntax. The rules for constructing a command or statement.

syntax checking. A function of the system, a compiler, the BASIC interpreter, or SEU that checks individual statements for errors in the structure of the statement.

system security. A system function that restricts the use of files, libraries, folders, and devices to certain users.

system value. Control information for the operation of certain parts of the system. A user can change the system value to define his working environment. System date and library list are examples of system values.

System/36 environment. A function of the operating system that processes most of the System/36 operator control language (OCL) statements and procedure statements to run System/36 application programs and

allows the user to process the control language (CL) commands. Contrast with *System/38 environment*.

System/38 environment. A function of the operating system that processes most of the System/38 control language (CL) statements and programs to run System/38 application programs. Contrast with *System/36 environment*.

test library. A user-defined library used for debugging operations that does not contain objects needed for normal processing. Contrast with *production library*.

translation table. An object that contains a set of hexadecimal characters used to translate one or more characters of data. For example, unprintable characters can be translated to blanks, and lowercase alphabetic characters can be translated to uppercase characters. The system-recognized identifier for the object type is *TBL.

uninterruptible power supply. A source of power from a battery installed between the commercial power and the system that keeps the system running, if a commercial power failure occurs, until it can complete an orderly end to system processing.

unlink. To remove the association between a data file on disk and a file definition in a data dictionary. Contrast with *link*.

update authority. A data authority that allows the user to change the data in an object, such as a journal, a message queue, or a data area. See also *add authority*, *delete authority*, and *read authority*.

use authority. An object authority that allows the user to run a program or display the contents of a file. Use authority combines object operational authority and read authority.

user profile. An object with a unique name that contains the user's password, the list of special authorities assigned to a user, and the objects the user owns. It is used by the system to verify the user's authorization to read or use objects, such as files or devices, to run the jobs on the system.

variable. A name used to represent data whose value can be changed while the program is running by referring to the name of the variable.

view. An alternative representation of data from one or more tables. A view can include all or some of the columns contained in the table or tables on which it is defined.

work station. A device used to transmit information to or receive information from a computer; for example, a display station or printer.

write operation. An output operation that sends a processed record to an output device or output file.

zoned decimal format. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the far right byte; bits 0 through 3 of all other bytes contain 1s (hex F). For example, in zoned decimal format, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Same as unpacked decimal format. Contrast with *packed decimal format*.

Index

A

access path 2-4
- arrival sequence 4-16
- controlling when rebuilt 16-7
- description 4-16
- dynamic select/omit 6-11
- floating point considerations 4-22
- for logical file record format 6-14
- journaling 16-5
- keyed sequence 4-16
- maintenance of 4-25
- saving and restoring 16-1
- select/omit 6-11
- sharing 6-12
- using existing specifications 4-22

ACCPTH (access path) parameter 8-3
add data authority 7-2
adding
- data base records, new 10-7
- logical file member to a file 6-19
- members, general considerations 13-1
- physical file member to a file 5-2

adding a member 13-1
adding records
- using OPNQRYF 9-32

advantages of data base integration 1-4
ALLOCATE parameter 5-2
allocating
- amount of storage for members 5-2
- contiguous storage for members 5-2

arranging data 1-4
- key fields 4-17
- using the OPNQRYF command 9-31

arranging records
- arrival sequence 4-16
- keyed sequence 4-16

arrival sequence access path 2-4
- description 4-16

attributes, data base file 15-1
AUT parameter 4-28

B

backup
- saving files 16-1

binary data type 4-9
both field usage 6-3

C

capabilities, file 7-4
categories of logical files 2-2
Change Logical File Member (CHGLFM) command 13-2
Change Physical File Member (CHGPFM) command 13-2
changing
- data type of a field 6-2
- fields
  - length of 6-2
  - level checking, data base 14-1
  - logical files 14-5
  - physical files 14-2
- length of a field 6-2
- logical file member 13-2
- physical file member 13-2
- records using OPNQRYF 9-32
- source date/time 17-9
- source file 17-8

character data type 4-9
checking changes to the record format 8-5
CHGLFM (Change Logical File Member) command 13-2
CHGPFM (Change Physical File Member) command 13-2
CL run-time parameters 8-15
clearing
- physical file members 13-3

close a data base file 11-1
close considerations
- sequential-only processing 8-15
- shared files 8-9

CMP (compare) keyword 6-8
COBOL/400 program, data base parameters 8-15
collating sequence
- floating point 4-23

COMMIT parameter 8-4
commitment control 16-3
commitment control parameter 8-4
comparison of immediate, delay, and rebuild maintenance 4-27
CONCAT keyword 6-5
concatenated field
- length 6-4
- sign 6-5

concurrency considerations for data base file 8-5
CONTIG parameter 5-2
contiguous storage, allocating for members 5-2
control language 4-23
controlling how the system runs
- OPNQRYF command 9-29

converting data types in logical files 6-2
copying files
CPYF (Copy File) command 17-4
CPYSRCF (Copy Source File) command 17-4
CPYF (Copy File) command 17-4
CPYSRCF (Copy Source File) command 17-4
Create Logical File (CRTLF) command
with member specified 6-20
with one record format 6-1
with two record formats 6-14
Create Physical File (CRTPF) command
example 5-1
explanation of parameters 5-1
Create Source Physical File (CRTSRCPF) command
example 17-1
creating
field reference file 4-11
logical file
general description 6-13
with a key field 6-13
with more than one record format 6-14
methods 4-23
object using a source file 17-6
physical file
general description 5-1
using a data base source file 17-6
using an inline data file 17-7
source file 17-1
cross-reference information 15-1
CRTLF (Create Logical File) command
example
with member 6-20
with one record format 6-1
with two record formats 6-14
CRTPF (Create Physical File) command
example 5-1
explanation of parameters 5-1
CRTSRCPF (Create Source Physical File) command
example 17-1

D

DATA (Data) command
example 17-6
use 17-6
data authorities 7-2
data base attributes 15-1
data base cross-reference information 15-1
data base file
access path
arrival sequence 4-16
keyed sequence 4-16
maintenance of 4-25
sharing 6-12
adding members to 13-1
adding records 10-7
capabilities 7-4
closing a member 11-1
data base file *(continued)*
commitment control 16-3
concurrency considerations 8-5
creating 4-23
data authorities 7-2
data location 4-24
deleted records, using 10-10
deleting records 10-9
description 2-1
displaying attributes 15-1
displaying records 13-5
distributed data management 3-5
documentation 4-29
errors detected during processing, handling 12-1
estimating sizes A-1
field attributes 15-1
format selector 10-7
journaling data 16-2
level checking 14-1
locking considerations C-1
locks 8-5
maximum sizes A-1
minimum sizes A-2
open parameters 9-1
opening 9-1
opening query 9-2
positioning 10-1
public authority 7-3
reading records with arrival sequence access path 10-2
reading records with keyed sequence access path 10-3
record format 3-2
recovery
after abnormal end 16-8
recovery planning 16-1
releasing record locks 8-6
remote file access 3-5
removing members from 13-2
renaming members in 13-2
saving 16-1, 16-8
security 7-1
sequential only processing 8-13
sharing 8-7
sharing record formats 4-14
source 17-3
updating records 10-6
using externally described 4-3
using in programs 3-3
wait time 4-28, 8-6
data base management system
arranging data 1-4
characteristics 1-1
data definition 1-4
data independence 1-2
data integrity 1-3
data security 1-2
data sharing 1-2

data base management system *(continued)*
joining files 1-3
selecting data 1-4
data definition 1-4
data description specifications form 4-3
data dictionary 3-1
data independence 1-2
data integrity 1-3
data mapping 6-3
data members, selecting 6-18
data recovery 16-2
data security 1-2
data sharing 1-2
data type
defining 4-9
data types (DDS) 4-9
DBCS (double-byte character set) considerations B-1
DDM (distributed data management)
OPNQRYF considerations 9-31
DDS (data description specifications) 4-3
decimal positions 4-9
field level 4-6
file level 4-6
key field level 4-6
name restrictions 4-6
record level 4-6
select/omit field level 4-6
decimal positions entry (DDS) 4-9
default records
description 13-2
default value
for missing records in join 6-25
definition of
access path 2-4
arrival sequence access path 2-4
data base 1-1, 2-1
data base file 2-1
data base management system 1-1
data dictionary 3-1
dictionary described 3-1
field 2-1
field definition 2-1
file 2-1
file descriptions 2-1
keyed sequence access path 2-4
logical file 2-2
member 2-3
physical file 2-1
record 2-1
record format 2-1
tables 2-1
delay maintenance of keyed sequence access paths 4-25
delaying end of file processing 8-3
delete data authority 7-2
deleted records
description 13-2
locks, releasing or displaying 10-10
using 10-10
deleted records percentage parameter 5-3
deleting records 10-9
deleting 10-9
using OPNQRYF 9-32
derived fields 6-4
using OPNQRYF 9-22
DESCEND DDS keyword 4-17
descending key field sequence 4-17
describing
access paths 4-5
data base file 3-1, 4-3
logical files 6-1
record format 4-4
DFT keyword
for physical files 6-24
dictionary-described data
description 4-2
disk unit, assigning members to 4-24
displaying
file and member attributes 15-1
file field descriptions 15-1
program references 15-2
records 13-5
distributed data management 3-5
divide by zero errors
handling in OPNQRYF 9-24
DLTPCT (delete percentage) parameter 5-3
documentation aids
displaying
file field description 15-1
program references 15-2
OUTFILE parameter 15-4
save/restore status 17-9
source file 17-10
using DDS 4-11
double-byte character set (DBCS) considerations B-1
DTAMBRS (data members) parameter 6-18
duplicate key values
description 4-18
preventing 4-19
retrieval order 4-20
dynamic select/omit 6-11

E

editing (DDS) 4-11
EDTCDE keyword 4-11
EDTWRD keyword 4-11
end-of-file
EOFDLY parameter on OVRDBF command 10-4
reading sequentially after 10-4
entering source using SEU 17-3
EOFDLY parameter 8-3
errors
detected during processing, handling 12-1
using OPNQRYF 9-37
estimating file size A-1

examples
arranging records using OPNQRYF 9-31
changing
data type of a field 6-2
length of a field 6-2
concatenating fields 6-4
controlling how the system runs OPNQRYF 9-29
creating
data base source file 17-1
logical file with a key field 6-1
logical file with two record formats 6-14
physical file using a data base source file 17-6
physical file with arrival sequence access path 4-7
deriving fields using OPNQRYF 9-22
describing a physical file 4-7
displaying
file field descriptions 15-1
program references 15-2
divide by zero errors in OPNQRYF 9-24
duplicate key values 4-18
dynamically joining files using OPNQRYF 9-16
final total-only processing in OPNQRYF 9-28
floating point fields 6-6
FORMAT keyword, use of 6-2
format selector 10-8
handling missing record in secondary files using OPNQRYF 9-20
merging record formats using key fields 6-17
ordering records
using more than one key field 4-18
using one key field 4-18
PFILE keyword, use of 6-1
random processing using OPNQRYF 9-33
renaming fields 6-6
reordering fields for a logical file 6-2
selecting and omitting records for a logical file 6-9
selecting records using OPNQRYF 9-8
setting up a data base file 3-2
sharing
access path 6-12
record format 4-14
substring fields 6-6
summarizing data in OPNQRYF 9-25
translating fields 6-6
unique key values 4-19
unique-key processing using OPNQRYF 9-21
EXPCHK parameter 8-5
EXPDATE parameter 8-5
externally described data 4-1
externally described file 4-1

F

field 2-1
changing in
data base file 14-1
data type 6-2
length of 6-2
level checking, data base 14-1
logical files 14-5
physical files 14-2
concatenated 6-4
defining data base 4-7
floating point 6-6
length 4-9
renaming 6-6
substring 6-6
translating 6-6
field level DDS 4-6
field level described data 4-1
field reference file 4-11
field usage 6-3
both fields 6-3
input only fields 6-3
neither fields 6-4
FIFO (first in, first out) keyword 4-20
file level DDS 4-6
file name 8-2
file name parameter 4-23
FILE parameter 4-23, 8-2
file type parameter 4-24
FILETYPE (file type) parameter 4-24
final total-only processing
using OPNQRYF 9-28
floating point data type 4-9
floating point fields 6-6
collating sequence 4-23
data base considerations 6-6
using in access paths 4-22
force access path parameter 4-25
force write ratio parameter 4-24
force-end-of-data operation 10-9
FORMAT keyword
example 4-15, 6-2
restrictions 4-15
use 4-15, 6-2
format selector 10-7
FRCACCPTH parameter 4-25, 8-4
FRCRATIO parameter 4-24, 8-4, 16-3

G

grouping data
handling in OPNQRYF 9-25

H

high-level language
handling file errors 12-1
OPNQRYF considerations 9-31

I

I/O considerations
sequential-only processing 8-14
shared files 8-9
IDDU (interactive data definition utility) 4-3, 4-23
ignoring
access paths 8-3
record formats 8-4
immediate maintenance of keyed access paths 4-25
INHWRT (Inhibit Write) parameter 8-5
initializing
physical file members 13-2
inline data file 17-6
input only field usage 6-3
integrated data base system 1-4
interactive data definition utility (IDDU) 4-3

J

JDFTVAL keyword 6-20, 6-42
JDUPSEQ keyword 6-32
join field
description 6-21
different attributes 6-33
join specifications 6-22
redefining 6-34
using more than one 6-30
join logical file 2-2
data integrity considerations 6-46
DDS for 6-22
description 6-21
example, complex 6-44
example, simple 6-21
handling duplicate records in secondary files 6-32
joining a physical file to itself 6-40
joining 3 or more physical files 6-38
key fields 6-37
missing records from secondary file 6-42
neither fields 6-35
performance considerations 6-45
primary file 6-23
random access 6-28
reading 6-23
restrictions 6-21
rules summary 6-46
secondary file 6-23
select/omit statements 6-38
setting up 6-29
using default data 6-42
using more than one 6-30
joining files 1-3
using OPNQRYF 9-16
journaling access paths 16-5
journaling data base files 16-2

K

key field
controlling sequence 4-17
description 4-16
duplicate values 4-18
retrieval order of duplicate keys 4-20
unique values 4-19
using *NONE 6-17
using more than one 4-18
with OPNQRYF 9-15
key field level DDS 4-6
keyed sequence access path 2-4
description 4-16
maintenance of 4-25

L

length of a concatenated field 6-4
length of field 4-9
level check parameter 4-25
level checking
data base file 14-1
using OPNQRYF 9-36
LIFO (last in, first out) keyword 4-20
locked records 8-5
logical file 2-2
adding a member to 6-19
authority 7-2
concatenating fields 6-4
creating 6-13
data mapping 6-3
data members, selecting 6-18
DDS example 6-13
definition of 2-2
derived fields 6-4
describing 4-10
effect of changing fields in 14-5
estimating size A-2
field usage 6-3
floating point fields 6-6
member
adding to a file 6-19
changing the attributes 13-2
description 6-19
selecting data 6-18
omit function 6-7
record format
how to specify 6-1
renaming fields 6-6
select function 6-7
substring fields 6-6
translated fields 6-6
using to secure data 7-4
with multiple record formats 6-14

logical file categories 2-2

M

maintenance of a keyed sequence access path 4-25
mapped fields
using OPNQRYF 9-22
maximum data base file sizes A-1
maximum members parameter 4-24
MAXMBRS parameter 4-24
MBR (member) parameter 4-23
MBR parameter 8-2
example 6-20
member 2-3
adding to a file 13-1
allocating amount of storage for 5-2
allocating contiguous storage for 5-2
assigning to a disk unit 4-24
clearing physical file 13-3
closing data base file 11-1
initializing physical file 13-2
locks 8-7
logical file 6-19
maintenance of keyed sequence access path 4-25
maximum number of records in 5-2
member (MBR) parameter 4-23
name 8-2
opening 9-1
physical file 5-1
removing from a file 13-2
renaming in a file 13-2
reorganizing 13-3
saving and restoring 16-1
selecting data 6-18
source file 17-2
merging records using key fields 6-17
methods
creating a data base file 4-23
describing data 4-3
minimum file sizes A-2
missing record in join files 6-25
handling using OPNQRYF 9-20
model output files
all commands 15-4
DSPFD command 15-7
DSPJRN command 15-8
office commands D-1
multiple format logical file 2-2

N

name restrictions 4-6
naming a member the same as the file 13-1
national language support B-1
neither fields
example 6-35
usage 6-4

O

object
creating, using source files 17-6
ODP (open data path) 9-1
omit function
for logical file 6-7
open considerations
sequential-only processing 8-13
sharing files in the same job 4-28, 8-7
Open Data Base File (OPNDBF) command 9-1
open data path (ODP) 9-1
Open Query File (OPNQRYF) command
arranging records 9-31
common errors 9-37
DDM file considerations 9-31
deriving fields 9-22
divide by zero errors 9-24
dynamic keyed access path 9-15
dynamic record selection 9-8
dynamically joining files 9-16
examples 9-7
field use considerations 9-35
final total-only processing 9-28
FORMAT parameter considerations 9-30
group processing 9-25
handling missing records in secondary files 9-20
high-level language considerations 9-31
high-level language program considerations 9-31
level checking 9-36
mapped field definitions 9-22
missing join secondary control function 9-20
optimization option 9-29
override considerations 9-36
performance considerations 9-33
random processing 9-33
record sequencing considerations 9-31
selecting records 9-8
sharing files in a job 9-36
summarizing data 9-25
unique-key processing 9-21
using a different record format 9-5
using an existing record format 9-3
using file for more than input 9-32
using for more than input 9-32
using key fields 9-15
optimizing how the system runs
OPNQRYF command 9-29
OPTION parameter 8-2
order of specification in DDS
logical file 4-10
physical file 4-22
OUTFILE parameter 15-4
output file 15-4

P

packed decimal data type 4-9
performance considerations
  join logical file 6-45
  OPNQRYF command 9-29, 9-33
PFILE keyword
  example with one record format 6-1
  example with two record formats 6-14
physical file 2-1
  adding a member to 5-2
  adding fields 14-2
  authority 7-2
  creating 5-1
  data mapping 6-3
  DDS example 4-8
  defining with DDS 4-7
  definition of 2-1
  effect of changing fields 14-2
  estimating size A-2
  member
    adding to a file 5-2
    changing the attributes 13-2
    clearing 13-3
    description 5-1
    initializing 13-2
    reorganizing 13-3
  record format 4-5
POSDBF (Position Data Base File) command 10-2
POSITION parameter 8-3
positioning, file 10-1
preventing data from being changed 8-5
preventing duplicate key values 4-19
primary file, in join 6-22
processing data base file
  reading records with keyed sequence access path 10-3
processing data base files
  adding 10-7
  adding records 10-7
  closing 11-1
  deleting records 10-9
  force-end-of-data operation 10-9
  format selector program 10-7
  opening 9-1
  processing options 8-2
  RCLRSC command 11-1
  reading records with arrival sequence access path 10-2
  releasing record locks 10-6
  sequential-only processing 8-14
  setting file position 10-1
  shared files 8-9
  updating records 10-6
program described data 4-1
program described file 4-1
programs
  run-time parameters 8-15
  using data base files in 3-3
public authority 7-3

Q

query file
  arranging records 9-31
  considerations for field use 9-35
  controlling how the system runs 9-29
  DDM considerations 9-31
  deriving fields 9-22
  divide by zero errors 9-24
  dynamically joining files 9-16
  final total-only processing 9-28
  handling missing records in secondary files 9-20
  high-level language considerations 9-31
  level checking 9-36
  performance considerations 9-33
  random processing 9-33
  selecting records 9-8
  sharing files in a job 9-36
  summarizing data 9-25
  typical errors using 9-37
  unique-key processing 9-21
  using a different record format 9-5
  using an existing record format 9-3
  using file for more than input 9-32
  using key fields 9-15

R

random processing
  using OPNQRYF 9-33
RANGE keyword 4-11
RCDFMTLCK parameter 8-7
RCDLEN (record length) Parameter 5-3, 8-4
RCLRSC (Reclaim Resources) command 11-1
read data authority 7-2
reading with arrival sequence access path 10-2
reading with keyed sequence access path 10-3
rebuild maintenance of keyed access paths 4-25
rebuilding an access path for recovery 16-5
reclaim resources command 11-1
record 2-1
record format 2-1, 8-7
  changes to 8-5
  describing 4-4
  locks 8-7
  LVLCHK (level check) parameter 4-25
  number of fields 6-2
  sharing 4-14
record length 5-3, 8-4
record level DDS 4-6
record level described data 4-1
record lock wait time 8-5
record locks
  displaying 8-6
  releasing 8-6
  wait time 8-5

record selection
using a logical file 6-7
using OPNQRYF 6-12
record wait time 4-28
recovery
access paths 16-5
commitment control 16-3
data base 16-8
forcing data to auxiliary storage 16-3
journaling data 16-2
planning 16-1
summary 16-10
REF (reference) keyword 4-11
REFACCPTH (reference access path) keyword 4-22
relating source and objects 17-7
relationship
physical file and logical file record formats 6-2
records and record format 6-2
releasing record locks 8-6, 10-6
remote file access 3-5
removing a member from a file 13-2
renamed fields 6-6
renaming a member 13-2
Reorganize Physical File Member (RGZPFM) command
example 13-3
reorganizing source files 17-8
restore
status information 17-9
restrictions
join logical file 6-21
sharing files in the same job 8-8
RPG/400 program, data base parameters 8-15
run-time parameters
checking changes to the record format 8-5
commitment control 8-4
delaying end of file processing 8-3
expiration date for data 8-5
file name 8-2
file wait time 8-6
ignoring record formats 8-4
ignoring the keyed access path 8-3
member name 8-2
preventing data from being changed 8-5
processing options 8-2
record format locks 8-7
record wait time 8-5
sequential-only processing 8-13
setting initial file position 8-3
sharing files in the same job 8-7
specifying record length 8-4
summary 8-15
writing data and access paths to auxiliary storage 8-4

S

saving and restoring
access paths 16-1
files 16-1
members 16-1
secondary file, in join 6-22
security 7-1
add authority 7-2
delete authority 7-2
object existence authority 7-1
object management authority 7-2
object operational authority 7-1
public authority 4-28, 7-3
read authority 7-2
update authority 7-2
using logical files 7-4
select/omit records
access path selection 6-11
dynamic access path selection 6-11
using a logical file 6-7
using OPNQRYF 6-12, 9-8
selecting data 1-4
SEQONLY (sequential-only) parameter 8-13
sequence numbers
in source files 17-4
resetting with RGZPFM 17-8
sequential-only processing
close considerations 8-15
input/output considerations 8-14
open considerations 8-13
setting initial file position 8-3
setting up
logical files 6-1
physical files 5-1
SHARE parameter 4-28, 8-7
sharing
access path specifications 4-22
access paths 6-12
files in the same job
close considerations 8-9
general considerations 8-7
I/O considerations 8-9
open considerations 8-8
using OPNQRYF 9-36
record formats 4-14
sign of a concatenated field 6-4
simple logical file 2-2
SIZE parameter 5-2
source file
changing 17-8
creating 17-1
date/time changes 17-9
definition of 2-2
description 17-1
file parameter 4-24
for online documentation 17-10
IBM-supplied 17-2
member 17-2

source file *(continued)*
member parameter 4-24
processing in a program 17-5
record format 17-3
relating to object 17-7
reorganizing 17-8
sequence numbers 17-5
using device files 17-4
using SEU to enter and change 17-3
using to create a object 17-6
SRCDAT field, source file record format 17-3
SRCDTA field, source file record format 17-3
SRCFILE (source file) parameter 4-24
SRCMBR (source member) parameter 4-24
SRCOPT parameter
RGZPFM command 13-3
SRCSEQ field, source file record format 17-3
storage location parameter 4-24
Structured Query Language/400 4-3, 4-23
view 2-2
substring fields 6-6
summaries
data base file maximums A-1
data base locks C-1
recovery options 16-10
rules for join logical files 6-46
run-time parameters 8-15
summarizing data
handling in OPNQRYF 9-25
SYSTEM parameter 4-29

T

test options
preventing data from being changed 8-5
TEXT
DDS keyword 4-11
parameter on create and change commands 4-29
transaction recovery 16-3
translated fields 6-6

U

unique key processing
example 4-20
using OPNQRYF 9-21
unique key value 4-19
UNIT parameter 4-24
update data authority 7-2
updating data base records 10-6
using OPNQRYF 9-32
updating source using SEU 17-8

V

validity checking (DDS) 4-11
VALUES keyword 4-11, 6-8
view 2-2

W

wait time
file 4-28, 8-6
record 4-28, 8-5
WAITFILE parameter 4-28, 8-6
WAITRCD (wait record) parameter 8-5
writing access paths to auxiliary storage 4-25, 8-4
writing data to auxiliary storage 4-24, 8-4

Z

zoned decimal data type 4-9

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your IBM-approved remarketer. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

☐ If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your IBM representative or your IBM-approved remarketer to request additional publications.

Name _____

Company or Organization _____

Address _____

_____

City State Zip Code

Phone No. _____

Area Code

No postage necessary if mailed in the U.S.A.

Cut Along Line

Fold and Tape **Please do not staple**

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS / PERMIT NO. 40 /ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

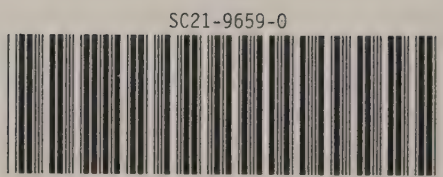
International Business Machines Corporation
Information Development
Department 245
Rochester, Minnesota, U.S.A. 55901

Fold and Tape **Please do not staple**

Cut Along Line

IBM®

Program Number
5728-SS1



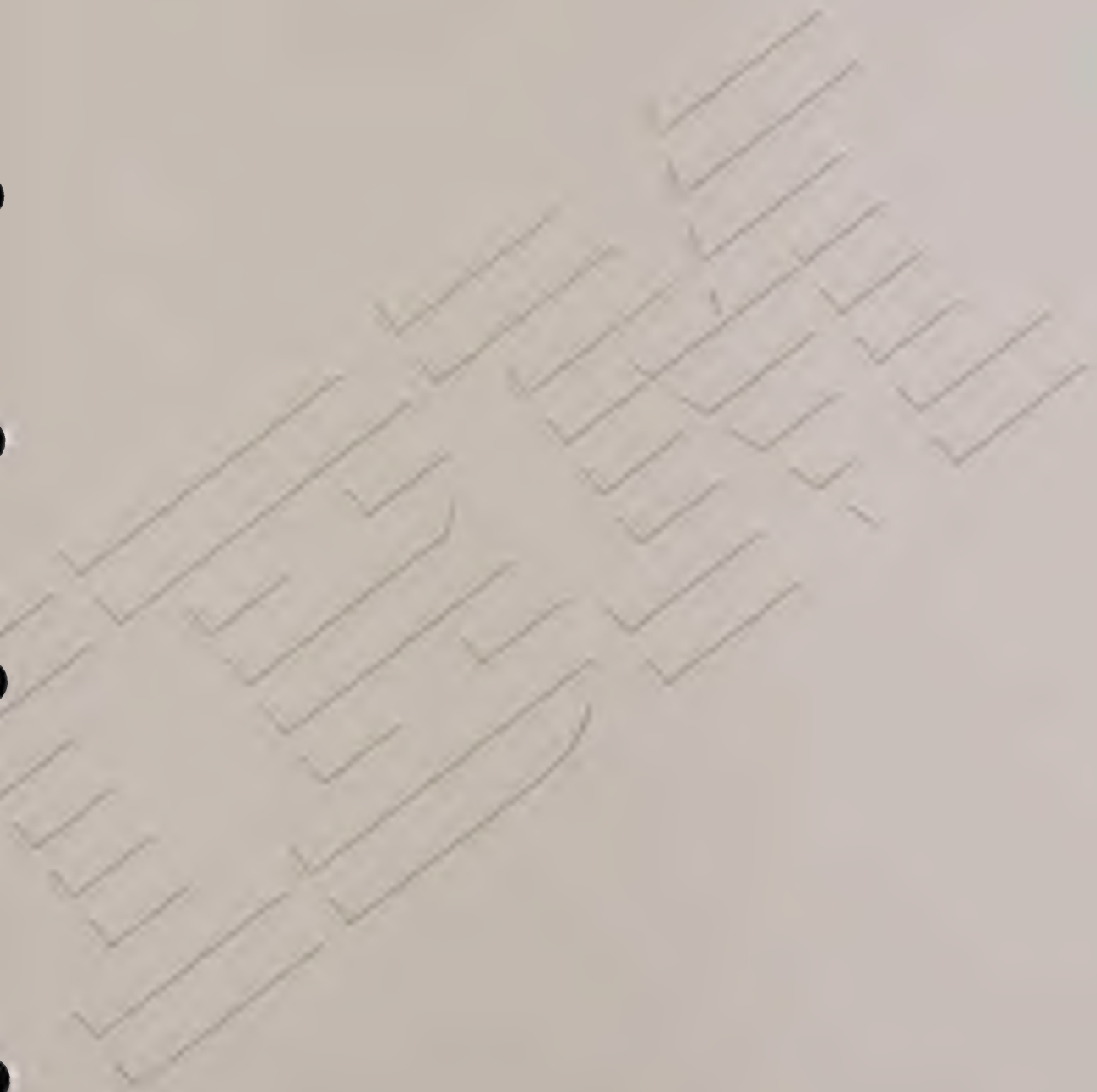
Printed in U.S.A.

IBM

AS/400™

SC09-1089-00

Languages: RPG/400 Reference



IBM

AS/400™

SC09-1089-00

Languages: RPG/400 Reference

First Edition (June 1988)

This edition applies to IBM AS/400™ RPG/400 (Program 5728-RG1), and to any subsequent releases and modifications until otherwise indicated in new editions. Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to:

IBM Canada Ltd.
Information Development
Department 849
1150 Eglinton Avenue East
North York, Ontario, Canada M3C 1H7

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

AS/400 is a trademark of International Business Machines Corporation, Armonk, NY.

© Copyright International Business Machines Corporation 1988.
IBM is a registered trademark of International Business Machines Corporation, Armonk, NY.

Contents

Chapter 1. Control Specifications 1-1
Control Specification Summary Chart 1-3
The Control Specification Statement 1-5
Position 6 (Form Type) 1-5
Positions 7-14 1-5
Position 15 (Debug) 1-5
Positions 16-17 1-5
Position 18 (Currency Symbol) 1-5
Position 19 (Date Format) 1-6
Position 20 (Date Edit) 1-6
Position 21 (Inverted Print) 1-6
Positions 22-25 1-7
Position 26 (Alternate Collating Sequence) 1-7
Positions 27-40 1-7
Position 41 (1P Forms Position) 1-7
Position 42 1-7
Position 43 (File Translation) 1-7
Positions 44-74 1-8
Positions 75-80 (Program Identification) 1-8

Chapter 2. File Description Specifications 2-1
Main File Description Line 2-2
Main File Description Line Summary Chart 2-3
Position 6 (Form Type) 2-6
Positions 7-14 (File Name) 2-7
Position 15 (File Type) 2-8
Input Files 2-8
Output Files 2-8
Update Files 2-8
Combined Files 2-8
Position 16 (File Designation) 2-8
Primary File 2-9
Secondary File 2-9
Record Address File 2-9
Array or Table File 2-9
Full Procedural File 2-9
Position 17 (End of File) 2-9
Position 18 (Sequence) 2-10
Position 19 (File Format) 2-11
Positions 20-23 2-11
Positions 24-27 (Record Length) 2-11
Position 28 (Mode of Processing) 2-11
Positions 29-30 (Length of Key Field or of Record Address Field) 2-12
Position 31 (Record Address Type) 2-12
Blank = Nonkeyed Processing 2-13
A = Character Keys 2-13
P = Packed Keys 2-13
K = Key 2-13

Position 32 (Type of File Organization or Additional Area) 2-14
Indexed Files 2-14
Nonkeyed Program Described File 2-14
Record Address File 2-14
Positions 33-34 (Overflow Indicator) 2-14
Positions 35-38 (Key Field Starting Location) 2-15
Position 39 (Extension Code) 2-15
Positions 40-46 (Device) 2-15
Positions 47-52 2-16
Position 53 (Continuation Lines) 2-16
Positions 54-59 (Name of Label Exit) 2-16
Positions 60-65 2-16
Position 66 (File Addition/Unordered) 2-16
Positions 67-70 2-17
Positions 71-72 (File Condition) 2-17
Positions 73-74 2-18
Positions 75-80 2-18
Continuation Line 2-18
Continuation Line Summary Chart 2-19
Continuation Line Options Summary Chart 2-21

Chapter 3. Extension Specifications 3-1
Extension Specification Summary Chart 3-2
Extension Specification Statement 3-4
Position 6 (Form Type) 3-4
Positions 7-10 3-4
Positions 11-18 (From File Name) 3-4
Positions 19-26 (To File Name) 3-5
Positions 27-32 (Array or Table Name) 3-7
Positions 33-35 (Number of Entries per Record) 3-7
Positions 36-39 (Number of Entries per Array or Table) 3-7
Positions 40-42 (Length of Entry) 3-8
Position 43 (Packed/Binary, Sign Left/Right) 3-8
Position 44 (Decimal Positions) 3-8
Position 45 (Sequence) 3-9
Positions 46-57 3-9
Positions 58-74 (Comments) 3-10
Positions 75-80 3-10

Chapter 4. Line Counter Specifications 4-1
Line Counter Specification Summary Chart 4-2
Line Counter Specification Statement 4-2
Position 6 (Form Type) 4-2
Positions 7-14 (File Name) 4-3
Positions 15-17 (Number of Lines Per Page) 4-3
Positions 18-19 (Form Length) 4-3
Positions 20-22 (Overflow Line Number) 4-3
Positions 23-24 (Overflow Line) 4-4
Positions 25-74 4-4
Positions 75-80 4-4

Chapter 5. Input Specifications 5-1
Input Specifications Summary Charts 5-3
Program Described Files, Record Identification Entries 5-3
Program Described Files, Field Description Entries 5-5
Externally Described Files, Record Identification Entries 5-6
Externally Described Files, Field Description Entries 5-7
Data Structure Statement Specifications 5-8
Data Structure Subfield Specifications 5-9
Program Described Files 5-10
Position 6 (Form Type) 5-10
Record Identification Entries 5-10
Positions 7-14 (File Name) 5-11
Positions 15-16 (Sequence) 5-11
Alphabetic Entries 5-11
Numeric Entries 5-11
Position 17 (Number) 5-12
Position 18 (Option) 5-12
Positions 19-20 (Record Identifying Indicator, or **) 5-12
Indicators 5-13
Look-Ahead Fields 5-13
Positions 21-41 (Record Identification Codes) 5-14
Positions 21-24, 28-31, and 35-38 (Position) 5-14
Positions 25, 32, and 39 (Not) 5-15
Positions 26, 33, and 40 (C/Z/D) 5-15
Positions 27, 34, and 41 (Character) 5-16
AND Relationship 5-16
OR Relationship 5-16
Position 42 5-16
Field Description Entries 5-16
Position 43 (Packed/Binary, Sign Left/Right) 5-17
Positions 44-51 (Field Location) 5-17
Position 52 (Decimal Positions) 5-18
Positions 53-58 (RPG/400 Field Name) 5-18
Positions 59-60 (Control Level) 5-18
Positions 61-62 (Matching Fields) 5-19
Positions 63-64 (Field Record Relation) 5-19
Positions 65-70 (Field Indicators) 5-20
Positions 71-74 5-21
Positions 75-80 5-21
Externally Described Files 5-21
Position 6 (Form Type) 5-21
Record Identification Entries 5-21
Positions 7-14 (Record Name) 5-21
Positions 15-18 5-22
Positions 19-20 (Record Identifying Indicator) 5-22
Positions 21-41 (Record Identification Code) 5-22
Positions 42-74 5-22
Positions 75-80 5-22
Field Description Entries 5-22
Positions 7-20 5-22
Positions 21-30 (External Field Name) 5-22

Positions 31-52 5-22
Positions 53-58 (RPG/400 Field Name) 5-23
Positions 59-60 (Control Level) 5-23
Positions 61-62 (Matching Fields) 5-23
Positions 63-64 5-23
Positions 65-70 (Field Indicators) 5-23
Positions 71-74 5-24
Positions 75-80 5-24
Data Structure Specifications 5-24
Data Structure Statement Specifications 5-24
Position 6 (Form Type) 5-24
Positions 7-12 (Data Structure Name) 5-24
Positions 13-16 5-24
Position 17 (Number) 5-25
Position 18 (Option) 5-25
Positions 19-20 (Record Identifying Indicator) 5-25
Positions 21-30 (External Name) 5-25
Positions 31-43 5-25
Positions 44-47 (Occurs N Times) 5-26
Positions 48-51 (Length) 5-26
Positions 52-74 5-26
Positions 75-80 5-26
Data Structure Subfield Specifications 5-26
Positions 7-20 5-26
Positions 21-30 (External Field Name) 5-26
Positions 31-42 5-27
Position 43 (Packed/Binary, Sign Left/Right) 5-27
Positions 44-51 (Field Location) 5-27
Position 52 (Decimal Positions) 5-27
Positions 53-58 (RPG/400 Field Name) 5-27
Positions 59-74 5-27
Positions 75-80 5-27

Chapter 6. Calculation Specifications 6-1
Calculation Specification Summary Chart 6-3
Calculation Specification Statement 6-6
Position 6 (Form Type) 6-6
Positions 7-8 (Control Level) 6-6
Control Level Indicators 6-6
Last Record Indicator 6-7
Subroutine Identifier 6-7
AND/OR Lines Identifier 6-7
Positions 9-17 (Indicators) 6-7
Positions 18-27 (Factor 1) 6-8
Literals 6-8
Positions 28-32 (Operation) 6-9
Positions 33-42 (Factor 2) 6-9
Positions 43-48 (Result Field) 6-9
Positions 49-51 (Field Length) 6-10
Position 52 (Decimal Positions) 6-10
Position 53 (Half Adjust) 6-11

Positions 54-59 (Resulting Indicators) 6-11
Positions 60-74 (Comments) 6-12
Positions 75-80 6-12

Chapter 7. Output Specifications 7-1
Output Specifications Summary Charts 7-3
Program Described Files, Record Identification and Control Entries (Record Line) 7-3
Program Described Files, Field Description and Control Entries (Field Line) 7-5
Externally Described Files, Record Identification and Control Entries . . . 7-6
Externally Described Files, Field Description and Control Entries 7-8
Program Described Files 7-8
Position 6 (Form Type) 7-8
Record Identification and Control Entries 7-9
Positions 7-14 (File Name) 7-9
Positions 14-16 (AND/OR) 7-9
Position 15 (Type) 7-10
Positions 16-18 (ADD/DEL) 7-10
Position 16 (Fetch Overflow/Release) 7-11
Fetch Overflow 7-11
Release 7-11
Positions 17-22 (Space and Skip) 7-11
Position 17 (Space Before) 7-12
Position 18 (Space After) 7-12
Positions 19-20 (Skip Before) 7-12
Positions 21-22 (Skip After) 7-13
Positions 23-31 (Output Indicators) 7-13
Positions 32-37 (EXCPT Name) 7-14
Field Description and Control Entries 7-15
Positions 23-31 (Output Indicators) 7-15
Positions 32-37 (Field Name) 7-15
Field Names, Blanks, Tables and Arrays 7-15
PAGE, PAGE1-PAGE7 7-16
*PLACE 7-16
UDATE, UDAY, UMONTH, UYEAR 7-16
*IN, *INxx, *IN,xx 7-16
Position 38 (Edit Codes) 7-16
Position 39 (Blank After) 7-17
Positions 40-43 (End Position in Output Record) 7-17
Position 44 (Packed/Binary, Sign Left/Right) 7-18
Positions 45-70 (Constant or Edit Word) 7-18
Constants 7-19
Edit Words 7-19
Format Name 7-19
Positions 71-74 7-19
Positions 75-80 7-19
Externally Described Files 7-19
Position 6 (Form Type) 7-19
Record Identification and Control Entries 7-19
Positions 7-14 (Record Name) 7-20
Positions 14-16 (AND/OR) 7-20

Position 15 (Type) 7-20
Position 16 (Release) 7-20
Positions 16-18 (ADD/DEL) 7-20
Positions 16-22 (Fetch Overflow/Space/Skip) 7-21
Positions 23-31 (Output Indicators) 7-21
Positions 32-37 (EXCPT Name) 7-21
Field Description and Control Entries 7-21
Positions 23-31 (Output Indicators) 7-21
Positions 32-37 (Field Name) 7-21
Position 38 (Edit Codes) 7-22
Position 39 (Blank After) 7-22
Positions 40-43 (End Position in Output Record) 7-22
Position 44 (Packed/Binary, Sign Left/Right) 7-22
Positions 45-70 (Constant or Edit Word) 7-22
Positions 71-74 7-22
Positions 75-80 7-22

Chapter 8. Auto Report Feature 8-1
Format of the Auto Report Generated Specifications 8-1
Order of Auto Report Generated Specifications 8-2
Calculation Specifications 8-3
Output Specifications 8-3
Comment Statements 8-3
Restriction 8-4
Options Specifications 8-4
Position 6 (Form Type) 8-5
Positions 7-26 8-5
Position 27 (Date Suppress) 8-5
Position 28 (*Suppress) 8-5
Positions 29-74 8-6
*AUTO Specifications 8-6
*AUTO Page Heading Specifications 8-7
Record Description Specifications 8-7
Positions 7-14 (File Name) 8-7
Position 15 (Type) 8-7
Position 16 8-7
Positions 17-22 (Spacing and Skipping) 8-7
Positions 23-31 (Output Indicators) 8-8
Positions 32-37 (*AUTO) 8-8
Positions 38-70 8-9
Field Description Specifications 8-9
Positions 7-31 8-9
Positions 32-37 (Field Name) 8-9
Position 38 (Edit Codes) 8-9
Position 39 (Blank After) 8-10
Positions 40-44 8-10
Positions 45-70 (Constant or Edit Word) 8-10
*AUTO Output Specifications 8-10
Record Description Specifications 8-11
Positions 7-14 (Filename) 8-11
Position 15 (Type) 8-11

Position 16 (Fetch Overflow) 8-11
Positions 17-22 (Spacing and Skipping) 8-12
Positions 23-31 (Output Indicators) 8-12
Positions 32-37 (*AUTO) 8-12
Positions 38-70 8-13
Field Description (Blank or B in Position 39) 8-13
Positions 7-22 8-13
Positions 23-31 (Output Indicators) 8-13
Positions 32-37 (Field Name) 8-13
Position 38 (Edit Codes) 8-14
Position 39 (Blank After) 8-14
Positions 40-43 (End Position in Output Record) 8-14
Position 44 8-14
Positions 45-70 (Constant) 8-15
Field Description (A in Position 39) 8-15
Generated Total Fields 8-16
Considerations 8-16
Positions 7-22 8-17
Positions 23-31 (Output Indicators) 8-17
Positions 32-37 (Field Name) 8-17
Position 38 (Edit Codes) 8-18
Position 39 8-18
Positions 40-43 (End Position in Output Record) 8-19
Position 44 8-19
Positions 45-70 (Constant) 8-19
Field Description (C in Position 39) 8-19
Positions 7-38 8-19
Position 39 8-19
Positions 40-44 8-19
Positions 45-70 (Constant) 8-19
Field Description (1-9 or R in Position 39) 8-20
Positions 7-31 8-21
Positions 32-37 (Field Name) 8-21
Position 38 (Edit Code) 8-21
Position 39 8-21
Positions 40-43 (End Position in Output Record) 8-22
Position 44 8-22
Positions 45-70 (Constant or Edit Word) 8-22
/COPY Statement 8-22

Chapter 9. Operation Codes 9-1
Arithmetic Operations 9-5
Move Operations 9-8
Move Zone Operations 9-9
Compare Operations 9-9
Declarative Operations 9-10
Subroutine Operations 9-10
SETON and SETOF Operations 9-11
Structured Programming Operations 9-11
Call and Branching Operations 9-13
File Operations 9-13

Information Operations 9-14
Message Operation 9-15
Data Area Operations 9-15
OPERATION CODES 9-16
ACQ (Acquire) 9-17
ADD (Add) 9-18
ANDxx (And) 9-19
BEGSR (Begin Subroutine) 9-20
BITOF (Set Bits Off) 9-21
BITON (Set Bits On) 9-22
CABxx (Compare and Branch) 9-24
CALL (Call a Program) 9-27
CASxx (Case) 9-29
CHAIN (Random Retrieval from a File Based on Record Number or Key Value) 9-31
CLOSE (Close Files) 9-34
COMIT (Commit) 9-36
COMP (Compare) 9-37
DEBUG (Debug Function) 9-39
Records Written for DEBUG 9-40
DEFN (Field Definition) 9-42
*LIKE DEFN 9-42
*NAMVAR DEFN 9-43
DELET (Delete Record) 9-46
DIV (Divide) 9-47
DO (Do) 9-48
DOUxx (Do Until) 9-50
DOWxx (Do While) 9-52
DSPLY (Display Function) 9-54
DUMP (Program Dump) 9-57
ELSE (Else Do) 9-58
END (End) 9-59
ENDSR (End Subroutine) 9-60
EXCPT (Calculation Time Output) 9-61
EXFMT (Execute Format) 9-66
EXSR (Execute Subroutine) 9-67
Coding Subroutines 9-67
FEOD (Force End of Data) 9-69
FORCE (Force A Certain File to Be Read on the Next Cycle) 9-70
FREE (Deactivate a Program) 9-71
GOTO (Branch To) 9-73
IFxx (If/Then) 9-75
IN (Retrieve a Data Area) 9-77
KFLD (Define Parts of a Key) 9-79
KLIST (Define A Composite Key) 9-80
LOKUP (Lookup) 9-82
MHHZO (Move High To High Zone) 9-85
MHLZO (Move High to Low Zone) 9-86
MLHZO (Move Low to High Zone) 9-87
MLLZO (Move Low to Low Zone) 9-88
MOVE (Move) 9-89

MOVEA (Move Array) 9-91
Character MOVEA Operations: 9-91
Numeric MOVEA Operations: 9-91
General MOVEA Operations 9-92
MOVEL (Move Left) 9-96
MULT (Multiply) 9-99
MVR (Move Remainder) 9-100
NEXT (Next) 9-101
OCUR (Set/Get Occurrence of a Data Structure) 9-102
OPEN (Open File for Processing) 9-105
ORxx (Or) 9-106
OUT (Write a Data Area) 9-107
PARM (Identify Parameters) 9-109
PLIST (Identify a Parameter List) 9-111
POST (Post) 9-113
READ (Read a Record) 9-114
READC (Read Next Modified Record) 9-117
READE (Read Equal Key) 9-118
READP (Read Prior Record) 9-120
REL (Release) 9-122
RETRN (Return to Caller) 9-123
ROLBK (Roll Back) 9-124
SETGT (Set Greater Than) 9-125
SETLL (Set Lower Limit) 9-129
SETOF (Set Off) 9-133
SETON (Set On) 9-134
SHTDN (Shut Down) 9-135
SORTA (Sort an Array) 9-136
SQRT (Square Root) 9-137
SUB (Subtract) 9-138
TAG (Tag) 9-139
TESTB (Test Bit) 9-140
TESTN (Test Numeric) 9-142
TESTZ (Test Zone) 9-144
TIME (Time of Day) 9-145
UNLCK (Unlock a Data Area) 9-146
UPDAT (Modify Existing Record) 9-147
WRITE (Create New Records) 9-149
XFOOT (Summing the Elements of an Array) 9-151
Z-ADD (Zero and Add) 9-152
Z-SUB (Zero and Subtract) 9-153

Chapter 10. RPG Words with Special Functions 10-1
User Date (UDATE, UMONTH, UDAY, UYEAR) 10-2
Rules for User Date 10-2
PAGE, PAGE1-PAGE7 10-3
Rules for PAGE, PAGE1-PAGE7 10-3
Figurative Constants (*BLANK/*BLANKS, *ZERO/*ZEROS, *HIVAL, *LOVAL, *ALL'X..') 10-4
Rules for Figurative Constants 10-5

Index X-1

About This Manual

Who Should Use this Manual

This manual is a reference for the RPG/400 programming language on the AS/400[1] system. It is intended for readers who, having read the *RPG/400 User's Guide*, are interested in more specific and more detailed explanations of RPG/400 code. Some references are made to the AS/400 system and its various products and features. In some instances, special reference is made to specific publications, but if no such reference is given, the reader should use the *RPG/400 User's Guide*.

The topics this manual covers include:

- RPG/400 specifications
- Auto Report feature
- Operation Codes
- RPG/400 words with special functions.

How This Manual is Organized

This manual is a reference to RPG/400. Chapter 1 to 7 describe the RPG/400 specifications in detail. Chapter 8 provides a detailed view of the Auto Report feature. Chapter 9 describes RPG/400 operations codes. Chapter 10 describes RPG/400 words with special functions.

What You Should Know

Before you use this manual, you should be familiar with certain information:

- You should know how to describe files to the system and how to create jobs and output queues. This information is contained in *Programming: Data Management Guide*, SC21-9658.
- You should be familiar with your display station (also known as a work station), and its controls. There are also some elements of its display and certain keys on the keyboard that are standard regardless of which software system is currently running at the display station, or which hardware system the display station is hooked up to. Some of these keys are:
  - Cursor movement keys
  - Command keys
  - Field exit keys
  - Insert and delete keys
  - The Error Reset key.

[1] AS/400 is a trademark of International Business Machines Corporation.

This information is contained in *System Operations: Display Station User's Guide* SC21-9744.

- You should know how to operate your display station when it is hooked up to the IBM AS/400 System and running AS/400 software. This means knowing about OS/400 and the Control Language (CL) to do such things as:
- Sign on and sign off the display station
- Interact with displays
- Use Help
- Enter control commands and procedure commands
- Call utilities
- Respond to messages.

To find out more about this operating system and its control language, refer to:

- *Programming: Control Language Reference*, SBOF-0481
- *Programming: Control Language Programmer's Guide*, SC21-8077
- *Programming: Command Reference Summary*, SC21-8076
- *Programming: System Reference Summary*, SC21-8104

- You should know how to call and use certain utilities available on the AS/400 System:
  - The Screen Design Aid (SDA) utility used to design and code displays. This information is contained in *Application Development Tools: Screen Design Aid User's Guide and Reference*, SC09-1171.
  - The Source Entry Utility (SEU), which is a full-screen editor you can use to enter and update your source and procedure members. This information is contained in *Application Development Tools: Source Entry Utility User's Guide and Reference*, SC09-1172.
- You should know how to interpret displayed and printed messages. This information is contained in *RPG/400 User's Guide*.
- You should be familiar with the RPG/400 program cycle, how indicators affect the program cycle, and how to code entries on the RPG/400 specification sheets.

If You Need More Information

You might need to refer to other AS/400 manuals for specific information about a particular topic. They are listed below:

- *Information Directory*, GC21-9678, which contains a brief description of each manual in the AS/400 library and information on how to order additional publications.
- *Licensed Programs Installation Guide*, SC21-9765, which describes how to install RPG/400 on your system.
- *System Operations: Operator's Guide*, SC21-8082, which describes how to operate the AS/400 System.
- *Programming: Data Description Specifications Reference*, SC21-9620, which describes data description specifications that are used for describing files.

- *Communications: Distributed Data Management User's Guide*, SC21-9600, which contains information about remote communication for the RPG/400 programmer.
- *Programming: Data Base Guide*, SC21-9659, which contains a detailed discussion of the AS/400 data base structure. This manual also describes how to use data description specifications (DDS) keywords.
- *Communications: Programmer's Guide*, SC21-9590, which provides information an application programmer needs to write applications that use AS/400 communications and the Intersystem Communications Function file.
- *Programming: Graphical Data Display Manager Programming Reference*, SC33-0537, and *Programming: Graphical Data Display Manager Programming Guide*, SC33-0536, which provide guidance on the Graphical Data Display Manager (GDDM)[2] for programmers who need to write graphics applications.
- *System/38 Environment Programmer's Guide and Reference*, SC21-9755, which describes migrating from System/38 and converting to AS/400 System.
- *Programming: Data Management Guide*, SC21-9658, which contains information about managing key aspects of the system.

---

2 GDDM is a trademark of International Business Machines Corporation.

RPG/400 Reference Manual

This manual contains reference information for RPG/400 language. For examples and discussion of the use of the code and specifications detailed in this manual see the *RPG/400 User's Guide*.

Chapters 1 to 7 of this reference describe the RPG/400 in detail.

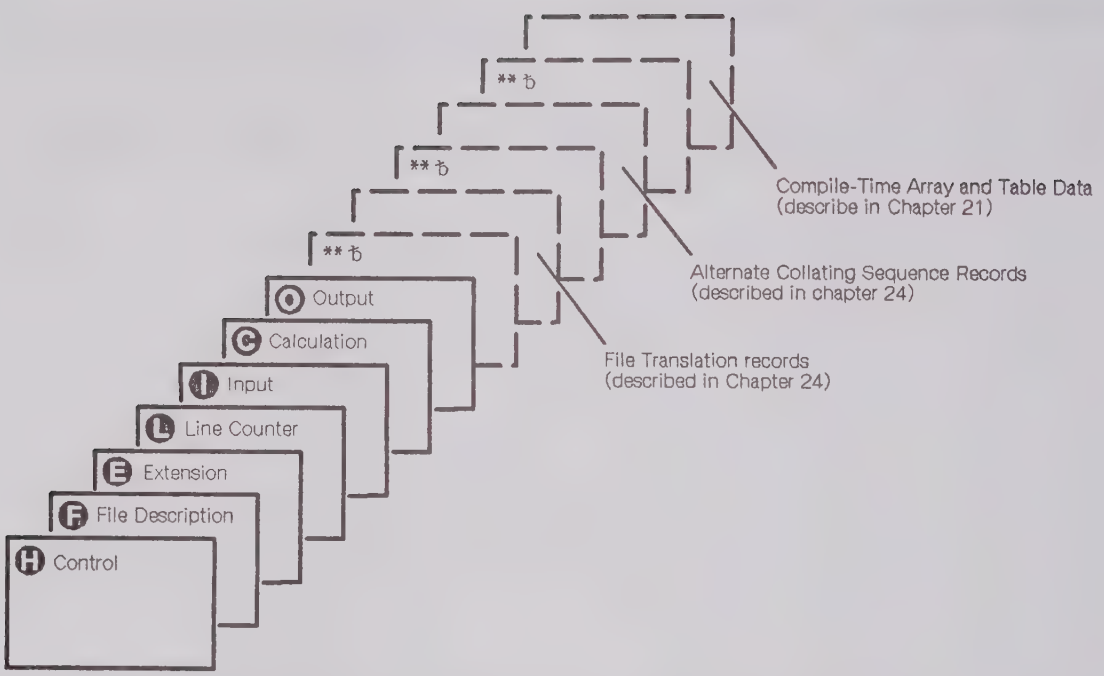
The remaining chapters in part 2 are:

- Chapter 8, "Auto Report Feature"
- Chapter 9, "Operation Codes"
- Chapter 10, "RPG Words with Special Functions"

An RPG/400 source program consists of several types of specifications that you code in a similar order in every RPG/400 program you write. Each specification has a fixed format, meaning that your entries on the specifications must be in the proper positions. This consistency and conformity in program coding simplifies learning and programming RPG/400.

The figure on the next page shows the RPG/400 specifications and their required order in a source program. RPG/400 does not require any of the specification types to be present, and any valid sequence of statements constitutes a program. The information you code on RPG/400 specifications can indicate the files your program uses, the input records, the operations to perform on these records, and the output.

A broken line is used to indicate an optional record.



H Control (header) specifications provide information about program generation and execution of the compiled program. Refer to Chapter 1, "Control Specifications" for a description of the entries on this specification.

F File description specifications define all files in the program. Refer to Chapter 2, "File Description Specifications" for a description of the entries on this specification.

E Extension specifications describe all arrays and tables and indicate how they are initialized. Refer to Chapter 3, "Extension Specifications" for a description of the entries on this specification.

L Line counter specifications indicate the length of the printer form and the number of lines to be printed on each page for each printer file in the program. Refer to Chapter 4, "Line Counter Specifications" for a description of the entries on this specification.

I Input specifications describe the records and fields in the input files and indicate how the records and fields are used by the program. Refer to Chapter 5, "Input Specifications" for a description of the entries on this specification.

C Calculation specifications describe calculations to be done by the program and indicate the order in which they are done. Calculation specifications can control certain input and output operations. Refer to Chapter 6, "Calculation Specifications" for a description of the entries on this specification.

O Output specifications describe the records and fields and indicate when they are to be written by the program. Refer to Chapter 7, "Output Specifications" for a description of the entries on this specification.

Note: The RPG/400 source program must be entered into the system in the order shown. Any of the specification types may be absent.

Figure 1-1. Order of the Types of Specifications in an RPG Source Program

Chapter 1. Control Specifications

The control specification statement provides information about program generation and running. This information is specified on the RPG/400 Control and File Description Specifications (see Figure 1-1 on page 1-2). You can create a data area named RPGHSPEC to contain the control specification to be used for all RPG/400 programs that do not contain a control specification. The data area must be a character string 80 positions long. Use the CL command CRTDTAARA (Create Data Area) to create the data area. Specify as the initial value of the data area the entries for the control specification that are to be used. For example, if the DEBUG operation is to be used for all RPG/400 programs, place an initial value of 1 in position 15 of the data area. (See the *Programming: Control Language Reference* for a description of the Create Data Area command.) The library in which the data area is placed must be specified in the library list when the RPG/400 program is compiled.

RPG uses the control specification that is present in the program. If a control specification is not present, RPG checks for the data area RPGHSPEC in *LIBL. If the data area is not found, RPG checks for the data area DFTHSPEC in QRPG. If it is not found, a default specification with blanks in positions 7 through 74 is used, (no data area is created). Note that since QRPG is the product library for the CRTRPGPGM command, if a data area called RPGHSPEC exists in QRPG it will always be found first, no matter where QRPG is on your library list. Use the data area DFTHSPEC in QRPG to create a common control specification for your installation, and use RPGHSPEC in your library to override this specification. See the description of the individual entries for the meaning of blank entries and for an explanation of the program name. If the default blank specification is used, asterisks are printed on the compiler listing under the Page/Line heading.

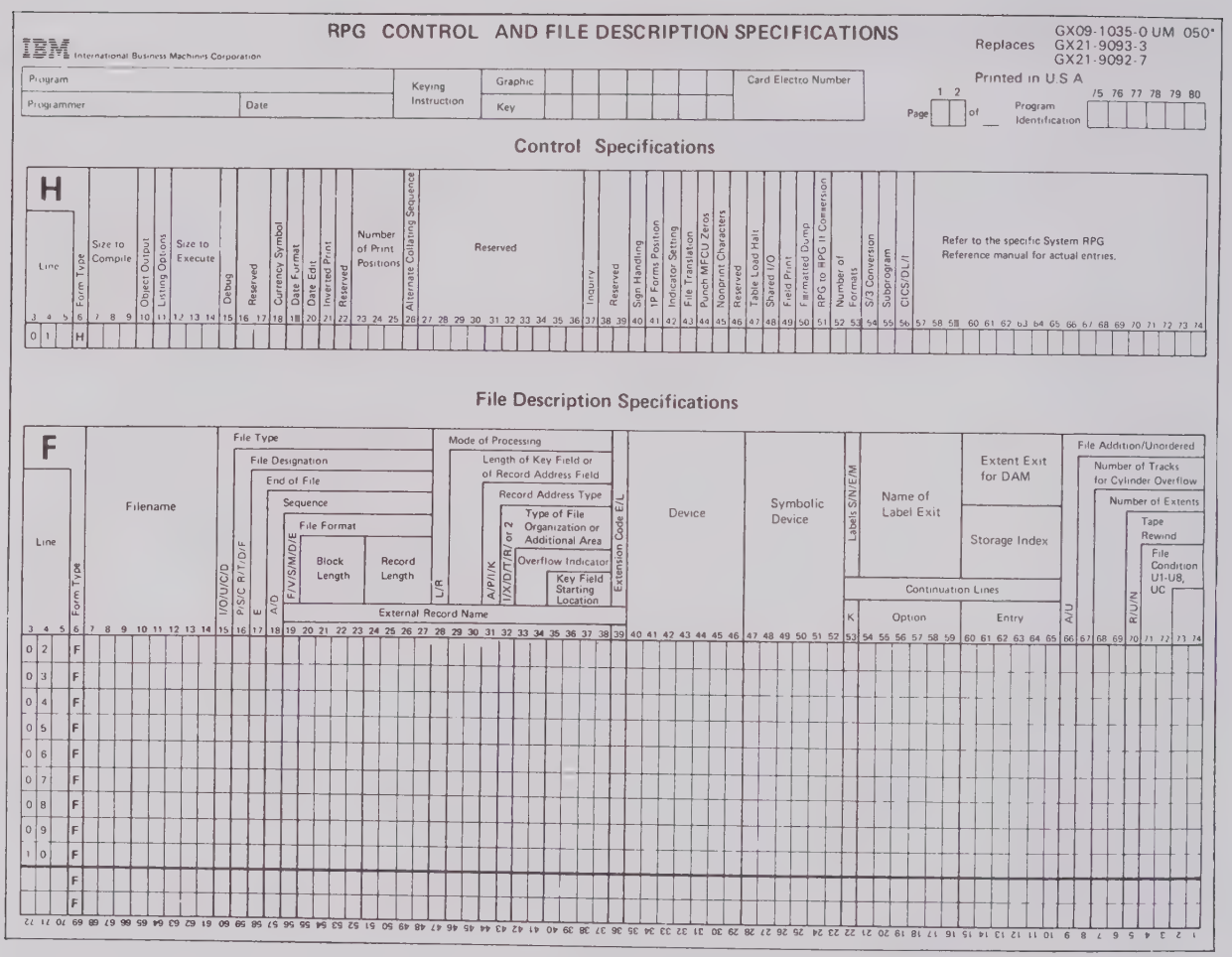


Figure 1-1. RPG Control and File Description Specifications Form

Control Specification Summary Chart

Table 1-1 (Page 1 of 2). Control Specification Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	H	Identification for a control (or header) specification.
7-14		Blank	
15	Debug	Blank	DEBUG/DUMP operation is not used. Compiler-generated symbols are not placed in the symbol table.
		1	DEBUG/DUMP operation is used. Compiler-generated symbols are placed in the symbol table.
16-17		Blank	
18	Currency symbol	Blank	Dollar sign ($) is used as currency symbol for editing.
		Currency symbol	Any character except zero (0), asterisk (*), comma (,), period (.), ampersand (&), minus sign (-), letter C, or letter R can be specified as the currency symbol.
19	Date format (UDATE)	Blank	Month/day/year format (mmddyy) if position 21 is blank. If position 21 contains a D, I, or J, the day/month/year (ddmmyy) format is used.
		M	Month/day/year (mmddyy). The separator character used depends on the entry in position 20 or position 21.
		D	Day/month/year (ddmmyy). The separator character used depends on the entry in position 20 or position 21.
		Y	Year/month/day (yymmdd). The separator character used depends on the entry in position 20 or position 21.
20	Date edit (Y edit code)	Any character	Separator character used between the fields of the date. If this position is blank, the separator is specified by the entry in position 21.
		&	Blank is used as separator character.

Table 1-1 (Page 2 of 2). Control Specification Summary Chart

Positions	Name	Entry	Explanation
21	Inverted print	Blank	Numeric fields and edit codes use a period and a comma for separators. If position 19 is blank, uses mmddyy format. If position 20 is blank, uses a slash (/) as separator for date.
		I	Numeric fields use comma as decimal notation and a period as a separator. If position 19 is blank, uses ddmmyy format. If position 20 is blank, uses a period (.) as separator for date.
		J	Numeric fields use comma as decimal notation and a period as a separator. If position 19 is blank, uses ddmmyy format. If position 20 is blank, uses a period (.) as separator for date.
		D	Numeric fields use a period as decimal notation and a comma as a separator. If position 19 is blank, uses ddmmyy format. If position 20 is blank, uses a slash (/) as separator for date.
22-25		Blank	
26	Alternate collating sequence	Blank	Normal collating sequence is used.
		S	Alternate collating sequence is used.
27-40		Blank	
41	1P forms position	Blank	First line is printed only once.
		1	First line can be printed repeatedly.
42		Blank	
43	File translation	Blank	No file translation is requested.
		F	Files are to be translated.
44-74		Blank	
75-80	Program identification		Entry used to assign a unique name to the program. This name can be overridden by the CRTRPGPGM command. If a name is not specified in positions 75 through 80 or on the CRTRPGPGM command, but the source file is a data base file, the member name is used as the program name. If the source is not from a data base file, the program name defaults to RPGOBJ.

The Control Specification Statement

Position 6 (Form Type)

An H must appear in position 6 to identify this line as the control specification (or header) statement.

Positions 7-14

Positions 7 through 14 must be blank.

Position 15 (Debug)

Entry	Explanation
Blank	DEBUG/DUMP operation is not done. Compiler-generated symbols are not placed in the symbol table.
1	DEBUG/DUMP operation is done. Compiler-generated symbols are placed in the symbol table.

Use position 15 to indicate whether the DEBUG/DUMP operation will be done. A 1 in position 15 when the source program is compiled causes the compiler to generate the object code for the DEBUG/DUMP operation.

The DEBUG entry also controls the contents of the symbol table that is produced with the program. If position 15 contains a 1, the compiler-generated symbols, starting with a period (.), are placed in the symbol table. If position 15 is blank, the compiler-generated symbols are not placed in the symbol table. The symbol table is printed with a program dump. You can then use the compiler-generated symbols in debugging a program.

See the DEBUG and DUMP operations in Chapter 9, "Operation Codes" for more information.

Positions 16-17

Positions 16 and 17 must be blank.

Position 18 (Currency Symbol)

Entry	Explanation
Blank	A dollar sign (hexadecimal 5B) is used as the currency symbol in editing (edit words and edit codes).
Currency Symbol	Any character except zero (0), asterisk (*), comma (,), period (.), ampersand (&), minus sign (-), the letter C, or the letter R may be specified as the currency symbol.

The specification of a currency symbol does not affect the user-defined edit codes 5 through 9.

Position 19 (Date Format)

The entry in this position specifies the format of the RPG/400 reserved word UDATE.

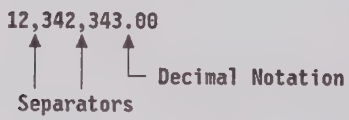
Entry	Explanation
Blank	Defaults to month/day/year if position 21 is blank. Defaults to day/month/year if position 21 contains a D, I, or J.
M	Month/day/year.
D	Day/month/year.
Y	Year/month/day.

Position 20 (Date Edit)

The entry in this position specifies the separator character to be used with the Y edit code. An entry of an ampersand (&) causes a blank to be used as the separator character. Any other character in position 20 is used as the separator character. If position 20 is blank, the separator character used for the Y edit code follows the specification in position 21 of the control specification.

Position 21 (Inverted Print)

The entry in this position specifies the notation for the UDATE format. It also specifies the decimal notation and the separator used for numeric literals and edit codes. The phrase *decimal notation* refers to the character that separates whole numbers from decimal fractions. The word *separator* refers to the character that separates the hundreds position from the thousands position, the hundred thousands position from the millions position, and so on. Below is an example of a number using a period for the decimal notation character and commas for the separator characters.



An entry in this position does not affect the edit words.

Entry	Explanation
Blank	Uses a period for the decimal notation, and a comma for the separator. If position 19 is blank, uses the month/day/year format for UDATE. If position 20 is blank, uses a slash as the separator for the Y edit code.
I	Uses a comma for the decimal notation, and a period for the separator. If position 19 is blank, uses the day/month/year format for UDATE. If position 20 is blank, uses a period as the separator for the Y edit code.
J	Uses a comma for the decimal notation, and a period for the separator. If position 19 is blank, uses the day/month/year format for UDATE. If position 20 is blank, uses a period as the separator for the Y edit code. When you use edit codes that cause zero balances to be

printed, zero is written to the left of the decimal notation (comma): 0,00. If the number of decimal positions in the field is equal to the length of the field, the decimal notation (comma) is the leftmost character printed.

D	Uses a period for the decimal notation, and a comma for the separator. If position 19 is blank, uses the day/month/year format for UDATE. If position 20 is blank, uses a slash as the separator for the Y edit code.

Positions 22-25

Positions 22 through 25 must be blank.

Position 26 (Alternate Collating Sequence)

Entry	Explanation
Blank	Normal collating sequence is used.
S	Alternate collating sequence is used.

Use position 26 to indicate whether an alternate collating sequence is to be used for character compare operations or match fields.

Positions 27-40

Positions 27 through 40 must be blank.

Position 41 (1P Forms Position)

Entry	Explanation
Blank	First line is printed only once.
1	First line can be printed repeatedly.

If the program contains more than one printer file, the entry in position 41 applies to each printer file that has 1P (first-page) output. This function may also be specified by the CL command OVRPRTF (Override to Print File) or in the printer device file.

Position 42

Position 42 must be blank.

Position 43 (File Translation)

Entry	Explanation
Blank	No file translation is requested.
F	Files are to be translated.

An entry of F indicates that a file translation table is to be used to translate all data in specified files.

Positions 44-74

Positions 44 through 74 must be blank.

Positions 75-80 (Program Identification)

The symbolic name entered in these positions is the name of the program that can be run. This name can be overridden by the PGM parameter in the CL command CRTRPGPGM (Create RPG Program)

If a name is not specified in positions 75 through 80 of the control specification or on the CRTRPGPGM command, but the source is from a data base file, the member name is used as the program name. If the source is not from a data base file, the program name defaults to RPGOBJ.

If the program name is specified on the control specification, the maximum length of the name is 6 characters. If the program name is specified in the CRTRPGPGM command, the maximum length of the name is 10 characters.

Note: Names entered here must follow the RPG symbolic name rules. Extended names are not allowed. However, the name specified on the CRTRPGPGM command can follow the extended naming rules.

Chapter 2. File Description Specifications

File description specifications identify each file used by a program. One file description specifications statement is required for each file in the program.

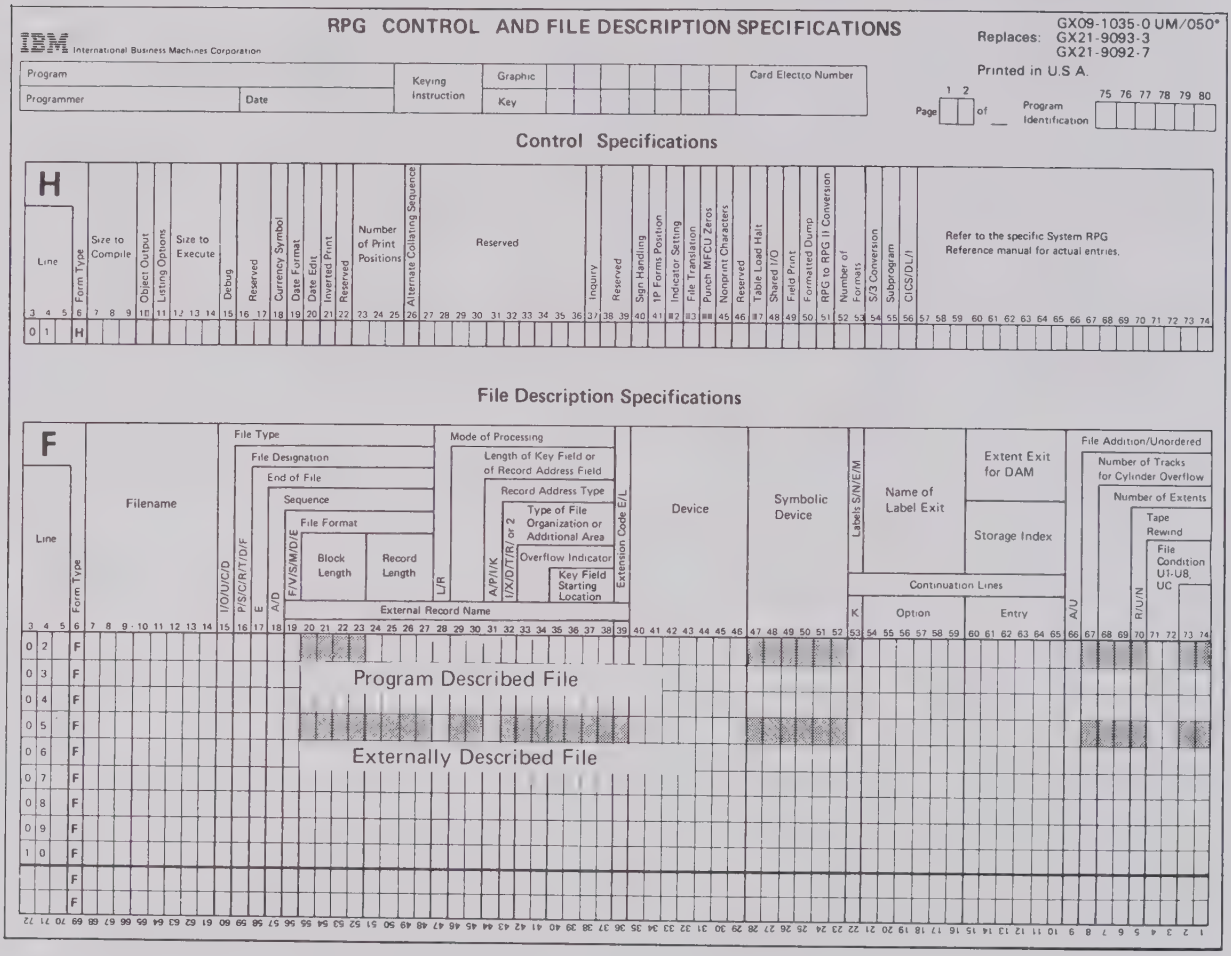
A maximum of 50 files can be described per program. Only one primary file can be specified; however, the presence of a primary file is not required. Only one record address file is allowed per program. A maximum of eight printer files can be specified. The maximum number of other file types is limited only by the maximum number of files allowed for the program.

Program described files require more entries on the file description specifications than externally described files. Many of the entries required for a program described file are part of the external description for an externally described file. Figure 2-1 on page 2-2 shows where entries can be made for a program described file and for an externally described file.

Enter the file description specifications on the RPG/400 Control and File Description Specifications (see Figure 2-1 on page 2-2).

You enter file description specifications on the main file description line. Additional specifications can be entered on the continuation lines.

Main File Description Line



Shaded positions must be blank.

Figure 2-1. RPG Control and File Description Specifications Form – Main File Description Line

Main File Description Line Summary Chart

Table 2-1 (Page 1 of 4). Main File Description Line Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	F	Identification for a file description specification.
7-14	File name	A valid file name	Every file must have a unique file name that is defined to OS/400. The file name can be from 1 to 8 characters long, and must begin with an alphabetic character.
15	File type	I	Input file.
		O	Output file.
		U	Update file.
		C	Combined (input/output) file.
16	File designation	Blank	Output file.
		P	Primary file.
		S	Secondary file.
		R	Record address file.
		T	Array or table file (prerun-time arrays or tables).
		F	Full procedural file.
17	End of file	E	All records from the file must be processed before the program can end. Not valid for files processed by a record address file.
		Blank	If position 17 is blank for all files, all records from all files must be processed before end-of-program (LR) can occur. If position 17 is not blank for all files, all records from this file may or may not be processed before end-of-program occurs in multifile processing.
18	Sequence	A or blank	Match fields are in ascending sequence.
		D	Match fields are in descending sequence.
19	File format	F	Program described file.
		E	Externally described file.

Table 2-1 (Page 2 of 4). Main File Description Line Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
20-23		Blank	
24-27	Record length	1-9999	Specifies the length of logical records contained in a program described file. The device record size restraints override an excessive record length.
28	Mode of processing	L	Sequential-within-limits processing by a record address file.
		Blank	Random or sequential processing. Random and sequential processing are implied by a combination of positions 16 and 31 of the file description specifications, and the calculation operation specified.
29-30	Length of key field or record address field	1-99	Length of the key field or the length of each entry in a record address file. Valid for program described files only. If the file being defined uses keys for record identification, enter the number of positions to be occupied by each record key. Record key length must be specified for indexed files.
		Blank	These positions must be blank for externally described files. For program described files, a blank entry indicates that keys are not used.
31	Record address type	Blank	• Relative record numbers are used to process the file. • Records are read consecutively. • Record address file contains relative record numbers. • Keys in record address limits file are in same format as keys in file being processed.
		A	Character keys (valid only for program described file specified as indexed file or as record address limits file).
		P	Packed keys (valid only for program described files specified as indexed file or record address limits file).
		K	Key values are used to process this file. Valid only for externally described files.

Table 2-1 (Page 3 of 4). Main File Description Line Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
32	Type of file organization or additional area	Blank	Program described file is processed without keys, or file is externally described.
		I	Indexed file. Valid only for program described files.
		T	Record address file that contains relative record numbers. Valid only for program described files.
33-34	Overflow indicators	Blank	No overflow indicator is used.
		0A-0G, 0V	Specified overflow indicator conditions the lines to be printed when overflow occurs. Valid only for program described printer files.
		01-99	Set on when overflow occurs. Valid for both program described and externally described printer files.
35-38	Key field starting location	Blank	Key fields are not used for this program described file, or the file is externally described.
		1-9999	Record position in a program described file in which the key field begins.
39	Extension code	Blank	No extension or line counter specifications are used.
		E	Extension specifications further describe the file.
		L	Line counter specifications further describe the file.
40-46	Device	PRINTER	The file is a printer file: the printer is used as output device.
		DISK	The file is a disk file: this device supports sequential and random read/write functions.
		WORKSTN	The file is a work station file: input/output is through a display or ICF file.
		SPECIAL	The file is a special file: input or output is associated with a device that is accessed by a user-supplied routine. The name of the routine must be specified in positions 54 through 59. The file must be a fixed unblocked format.
		SEQ	The file is a sequentially organized file: the actual device is specified outside the RPG program.
47-52		Blank	

Table 2-1 (Page 4 of 4). Main File Description Line Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
53	Continuation lines	Blank	This specification is not a continuation line. The following position explanations apply when position 53 is blank.
		K	Indicates a continuation line. For an explanation of positions 54-80 when position 53 contains K, see "Continuation Line" on page 2-18 on the following pages.
54-59	Name of label exit	Name of user- supplied routine	
60-65		Blank	
66	File addition/ unordered	Blank	No additions of records can be made to an input or update file. For output file, a blank is equivalent to A.
		A	Add records to a DISK file. Positions 16 through 18 of the output specifications must contain ADD, or a WRITE operation code must be used in the calculation specifications.
67-70		Blank	
71-72	File condition	Blank	The file can be used by the program. If it is an input file, it is opened by RPG/400.
		U1-U8	The file can be used by the program when the indicator is on; it is not used when the indicator is off.
		UC	Programmer control of first open. If a file is to be opened by an OPEN operation in the calculation specification, a UC entry causes the file not to be opened at program initialization. Not valid for primary, secondary, table, or record address input files, or for output files conditioned by 1P indicator.
73-74		Blank	
75-80		Optional	This space is available for comments.

Position 6 (Form Type)

An F must be entered in this position for file description specifications.

Positions 7-14 (File Name)

Entry	Explanation
A valid file name	Every file used in a program must have a unique name. The file name can be from 1 to 8 characters long, and must begin in position 7.

Each file used in the program is identified by a unique symbolic name in positions 7 through 14. symbolic names.)

The file name specified in positions 7 through 14 must be an existing file name that has been defined to OS/400, or one of the OS/400 override commands must be used to associate the RPG/400 file name to the file name defined to OS/400. For an externally described file, the file must exist at both compilation time and at run time. For a program described file, the file need exist only at run time. The file name specified in these positions, rather than the device name specified in positions 40 through 46, is used to point to the file. When the files are opened at run time, RPG/400 opens the files in the reverse order to that specified in the file description specifications. The RPG/400 device name defines the functions that can be processed on the associated file.

You can specify file names in positions 7 through 14 that correspond to predefined device file definitions supplied by IBM.

For program described files, the file name entered in positions 7 through 14 must also be entered on:

- Input specifications if the file is an input, an update, or a combined file
- Output specifications or an output calculation operation line if the file is an output, update, or combined file, or if the file is an input file and records are also being added to the file
- Extension specifications if the file is a table, array, or record address file, or a file processed by a record address file
- Calculation specifications if the file name is required for the operation code specified
- Line counter specifications if the device is a printer and default values are to be overridden in RPG/400.

For externally described files, the file name entered in positions 7 through 14 is the name used to locate the record descriptions for the file. The following rules apply to externally described files:

- Input and output specifications for externally described files are optional. They are required only if you are adding RPG/400 functions, such as control fields or record identifying indicators, to the external description retrieved.
- When an external description is retrieved, the record definition can be referred to by its record format name on the input, output, or calculation specifications.
- A record format name must be a unique symbolic name.

- A logical file with two record formats of the same name cannot have the duplicate format names renamed and cannot be externally described. However, RPG/400 can access such a file if it is program described.

Position 15 (File Type)

Entry	Explanation
I	Input file
O	Output file
U	Update file
C	Combined (input/output) file

Input Files

An input file is a file from which information is to be read by the program. An input file can contain data records, arrays, or tables, or can be a record address file.

Output Files

An output file is a file on which information is to be written.

Update Files

An update file is both an input file and an output file. The file is an update file if the program alters the data in one or more fields of any record contained in the file and writes that record back in the same place on the same file from which it was read. Only DISK files and SPECIAL files can be specified as update files.

Combined Files

A combined file is both an input file and an output file. When a combined file is processed, the output record contains only the data represented by the fields in the output record. This differs from an update file, where the output record contains the input record modified by the fields in the output record.

A combined file is valid for a SPECIAL or WORKSTN file. A combined file is also valid for a DISK or SEQ file if position 16 contains T (an array or table replacement file).

Position 16 (File Designation)

Entry	Explanation
Blank	Output file
P	Primary file
S	Secondary file
R	Record address file
T	Array or table file (prerun-time arrays or tables)
F	Full procedural file

Primary File

When more than one input file is used during cycle-controlled programming, the primary file is the controlling input file. When multiple input files that are designed as primary and secondary files are processed, only one input file must be specified as the primary file.

Secondary File

When more than one file is used during cycle-controlled programming, secondary files are input files that depend on other files. The processing of secondary files is determined by the order in which they are specified in the file description specifications and on the rules of multifile logic.

Record Address File

A record address file is a sequentially organized file used to select records from another file. Only one file in a program can be specified as a record address file. This file is described on the file description and extension specifications and never on the input specifications. The file processed by the record address file must also be specified on the extension specifications.

A record address file cannot be specified for the device SPECIAL. An externally described file cannot be specified as a record address file; however, a record address file can be used to process a program described file or an externally described file.

A record address file that contains relative record numbers must also have a T specified in position 32.

Array or Table File

Array and table files specified by a T in position 16 are loaded at prerun time. Leave this entry blank for array or table output files. SPECIAL cannot be specified as the device for array and table input files. An externally described file cannot be specified as an array or table file.

If a T is specified in position 16, a C may be specified in position 15 for a DISK or SEQ file. This C allows an array or table file to be read from or written to the same file (an array or table replacement file). The To and From file names on the extension specifications must specify this file name.

Full Procedural File

This entry is used when the input is controlled by calculation operations. File operation codes such as CHAIN or READ are used to do input functions.

Position 17 (End of File)

Entry	Explanation
E	All records from the file must be processed before the program can end. This entry is not valid for files processed by a record address file.
Blank	• If position 17 is blank for all files, all records from all files must be processed before end of program (LR) can occur.

- If position 17 is not blank for all files, all records from this file may or may not be processed before end of program occurs in multifile processing.

Use position 17 to indicate whether the program can end before all records from the file are processed. An E in position 17 applies only to input or update files specified as primary, secondary, or record address files.

If the records from all primary and secondary files must be processed, position 17 must be blank for all files or must contain Es for all files. For multiple input files, the end-of-program (LR) condition occurs when all input files for which an E is specified in position 17 have been processed. If position 17 is blank for all files, the end-of-program condition occurs when all input files have been processed.

When match fields are specified for two or more files and an E is specified in position 17 for one or more files, the LR indicator is set on after:

- The end-of-file condition occurs for the last file with an E specified in position 17.
- The program has processed all the records in other files that match the last record processed from the primary file.
- The program has processed the records in those files without match fields up to the next record with nonmatching match fields.

When no file or only one file contains match field specifications, no records of other files are processed after end of file occurs on all files for which an E is specified in position 17.

Position 18 (Sequence)

Entry	Explanation
A or blank	Match fields are in ascending sequence.
D	Match fields are in descending sequence.

Position 18 specifies the sequence of input fields used with the match fields specification (positions 61 and 62 of the input specifications). Position 18 applies only to files used as primary or secondary files. Use positions 61 and 62 of the input specifications to identify the fields containing the sequence information.

If more than one input file with match fields is specified in the program, a sequence entry in position 18 can be used to check the sequence of the match fields and to process the file using the matching record technique. The sequence need only be specified for the first file with match fields specified. If sequence is specified for other files, the sequence specified must be the same; otherwise, the sequence specified for the first file is assumed.

If only one input file with match fields is specified in the program, a sequence entry in position 18 can be used to check fields of that file to ensure that the file is in

sequence. By entering one of the codes M1 through M9 in positions 61 and 62 of the input specifications, and by entering an A or D in position 18, you specify sequence checking of these fields.

Sequence checking is required when match fields are used in the records from the file. When a record from a matching input file is found to be out of sequence, the RPG/400 exception/error handling routine is given control.

Position 19 (File Format)

Entry	Explanation
F	Program described file
E	Externally described file

An F in position 19 indicates that the records for the file are described within the RPG/400 program on input/output specifications.

An E in position 19 indicates that the record descriptions for the file are external to the RPG/400 source program. The compiler obtains these descriptions at compilation time and includes them in the source program.

An entry is required in position 19.

Positions 20-23

Positions 20 through 23 must be blank. (Block length, if allowed, is specified outside the RPG/400 program.)

Positions 24-27 (Record Length)

Use positions 24 through 27 to indicate the length of the logical records contained in a program described file. The maximum record size that can be specified is 9999; however, record-size restraints of any device override this value. This entry must be blank for externally described files.

Position 28 (Mode of Processing)

Entry	Explanation
L	Sequential-within-limits processing by a record address file
Blank	Sequential or random processing

Use position 28 to indicate whether the file is processed by a record address file that contains limits records.

A record address file used for limits processing contains records that consist of upper and lower limits. Each record contains a set of limits that consists of the lowest record key and the highest record key from the segment of the file to be processed. Limits processing can be used for keyed files specified as primary, secondary, or full procedural files.

The L entry in position 28 is valid only if the file is processed by a record address file containing limits records. Random and sequential processing of files is implied by a combination of positions 16 and 31 of the file description specifications, and by the calculation operation specified.

The operation codes SETLL and SETGT can be used to position a file; however, the use of these operation codes does not require an L in position 28.

Positions 29-30 (Length of Key Field or of Record Address Field)

Entry	Explanation
1-99	The number of positions required for the key field in a program described file or the length of the entries in the record address file (which must be a program described file). If the program described file being defined uses keys for record identification, enter the number of positions occupied by each record key. This entry is required for indexed files. If the keys are packed, the key field length should be the packed length; this is the number of digits in DDS divided by 2 plus 1 and ignoring any fractions. If the file being defined is a record address file, enter the number of positions that each entry in the record address file occupies.
Blank	These positions must be blank for externally described files. (The key length is specified in the external description.) For a program described file, a blank entry indicates that keys are not used.

Position 31 (Record Address Type)

Entry	Explanation
Blank	Relative record numbers are used to process the file. Records are read consecutively. Record address file contains relative record numbers. Keys in record address limits file are in the same format as keys in the file being processed.
A	Character keys (valid only for program described files specified as indexed files or as a record address limits file).
P	Packed keys (valid only for program described files specified as indexed files or as a record address limits file).
K	Key values are used to process the file. This entry is valid only for externally described files.

Blank = Nonkeyed Processing

A blank indicates that the file is processed without the use of keys, that the record address file contains relative Record numbers (a T in position 32), or that the keys in a record address limits file are in the same format as the keys in the file being processed.

A file processed without keys can be processed consecutively or randomly by relative record number.

Input processing by relative record number is determined by a blank in position 31 and by the use of the CHAIN, SETLL, or SETGT operation code. Output processing by relative record number is determined by a blank in position 31 and by the use of the RECNO keyword on the file description specifications.

A = Character Keys

The indexed file (I in position 32) defined on this line is processed by character record keys. (A numeric field used as the search argument is converted to zoned decimal before chaining.) The A entry must agree with the data format of the field identified as the key field (length in positions 29 and 30 and starting position in positions 35 through 38).

The record address limits file (R in position 16) defined on this line contains character keys. The file being processed by this record address file can have an A, P, or K in position 31.

P = Packed Keys

The indexed file (I in position 32) defined on this line is processed by packed decimal numeric keys. The P entry must agree with the data format of the field identified as the key field (length in positions 29 and 30 and starting position in positions 35 through 38).

Note: The sign of all decimal numeric input fields is forced to F or D. All numeric result fields specified by calculation specifications also have an F or D sign. Therefore, if the sign of the key field in the file is not F or D, a record-not-found error occurs when chaining to that file is specified.

The record address limits file defined on this line contains record keys in packed decimal format. The file being processed by this record address file can have an A, P, or K in position 31.

K = Key

A K entry indicates that the externally described file is processed on the assumption that the access path is built on key values. If the processing is random, key values are used to identify the records.

If this position is blank for a keyed file, the records are retrieved in arrival sequence.

Position 32 (Type of File Organization or Additional Area)

Entry	Explanation
Blank	The program described file is processed without keys, or the file is externally described.
I	Indexed file (valid only for program described files).
T	Record address file that contains relative record numbers (valid only for program described files).

Use position 32 to identify the organization of program described files.

Indexed Files

An indexed file can be processed:

- Randomly or sequentially by key.
- By a record address file (sequentially within limits). Position 28 must contain an L.

Nonkeyed Program Described File

A program described file that is processed without keys can be processed:

- Randomly by relative record numbers. Positions 28 and 31 must be blank.
- Consecutively. Positions 28 and 31 must be blank.
- As a record address file. Position 28 must be blank.

Record Address File

A record address file (indicated by an R in position 16) that contains relative record numbers must be identified by a T in position 32. (A record address file must be program described.) Each record retrieved from the file being processed is based on the relative record number in the record address file. (Relative record numbers cannot be used for a record address limits file.)

Each relative record number in the record address file is a 4-byte binary field; therefore, each four bytes of a record address file contains a relative record number. A minus one (-1 or hexadecimal FFFFFFFF) relative record number value stops the use of that record address file record. End of file occurs when all record address file records have been processed.

Positions 33-34 (Overflow Indicator)

Entry	Explanation
Blank	No overflow indicator is specified.
0A-0G, 0V	Specified overflow indicator conditions the lines to be printed when overflow occurs.
01-99	Set on when overflow occurs.

Indicators 0A through 0G, and 0V are not valid for externally described files.

Use positions 33 and 34 to specify an overflow indicator to condition which lines in each printer file will be printed when overflow occurs. This entry is valid only for a PRINTER device.

Only one overflow indicator can be assigned to a file. If more than one printer file in a program is assigned an overflow indicator, the indicator must be unique for each file.

Positions 35-38 (Key Field Starting Location)

Entry	Explanation
Blank	Key fields are not used for this program described file, or the file is externally described.
1-9999	Record position in a program described indexed file in which the key field begins.

Use positions 35 through 38 to identify the record position in which the key field for a program described indexed file begins. An entry must be made in these positions for a program described indexed file. The key field of a record contains the information that identifies the record. The key field must be in the same location in all records in the file. The entry in these positions must be right-adjusted. Leading zeros can be omitted.

Position 39 (Extension Code)

Entry	Explanation
Blank	No extension or line counter specifications are used.
E	Extension specifications further describe the file.
L	Line counter specifications further describe the file.

Use position 39 to indicate whether the program described file is further described on the extension specifications or on the line counter specifications. An E in position 39 applies only to array or table files or to record address files; an L in position 39 applies to files assigned to the PRINTER device.

Positions 40-46 (Device)

Entry	Explanation
PRINTER	File is a printer file: data output to the file is sent to a printer.
DISK	File is a disk file: this device supports sequential and random read/write functions. These files can be accessed on a remote system by DDM.
WORKSTN	File is a work station file: input/output is through a display or ICF file.

Entry	Explanation
SPECIAL	This is a special file: input or output is on a device that is accessed by a user-supplied routine. The name of the routine must be specified in positions 54 through 59. The file must be a fixed unblocked format.
SEQ	File is a sequentially organized file. The actual device is specified in a CL command or in the file description, which is accessed by the file name.

Use positions 40 through 46 to specify the RPG/400 device name to be associated with the file. On the AS/400 System the file name in positions 7 through 14, rather than the device name specified in positions 40 through 46, is used to point to the file. The RPG/400 device name defines the RPG/400 functions that can be done on the associated file. Certain functions are valid only for a specific RPG/400 device name, such as the EXFMT operation for WORKSTN. The file name specified in positions 7 through 14 can be overridden at compilation time or run time, therefore allowing you to change the input/output device used in the program.

Note that the RPG/400 device names are not the same as the system device names.

Positions 47-52

Positions 47 through 52 must be blank.

Position 53 (Continuation Lines)

A K in position 53 indicates a continuation line. See "Continuation Line" on page 2-18 for more information.

Positions 54-59 (Name of Label Exit)

When SPECIAL is the device entry (positions 40 through 46), the routine named in positions 54 through 59 handles the support for the special I/O device. The routine name is symbolic and must be left-adjusted. The name is used by the compiler to produce the linkage to the routine.

Positions 60-65

Positions 60 through 65 must be blank.

Position 66 (File Addition/Unordered)

Position 66 indicates whether records are to be added to a DISK file.

Entry	Explanation
Blank	No records can be added to an input or update file (I or U in position 15). For an output file (O in position 15), a blank is equivalent to an A.
A	Add records to the file. Positions 16 through 18 of the output record specifications for this file must contain ADD, or the WRITE operation code must be used in the calculation specifications.

See Table 2-2 on page 2-17 for the relationship between position 15 and position 66 of the file description specifications and positions 16 through 18 of the output specifications.

Table 2-2. Processing Functions for Files

Function	File Description Specifications Position 15	File Description Specifications Position 66	Output Specifications Positions 16-18
Create new file[1]	O	Blank	Blank
Add records to existing file[1]	O	A	ADD
Process file	I	Blank	Blank
Process file and add records to the existing file	I	A	ADD
Process file and update the records (update or delete)	U	Blank	Blank
Process file and add new records to an existing file	U	A	ADD
Process file and delete an existing record from the file	U	Blank	DEL

[1]Within RPG, the term *create a new file* means to add records to a file that has been created by a **create file** command. Thus, the first two entries in this table perform the identical function. Both are listed to show that there are two ways to specify that function.

Positions 67-70

Positions 67 through 70 must be blank.

Positions 71-72 (File Condition)

Entry	Explanation
Blank	The file can be used by the program, and, if it is an input file, the file is opened by RPG/400.
U1-U8	The file can be used by the program when the indicator is on; it is ignored when the indicator is off.
UC	Programmer control of first open. If a file is to be initially opened by the OPEN operation in the calculation specifications, then a UC entry causes the file not to be opened at program initialization. This entry is not valid for input files designated as primary, secondary, table, or record address files, or for output files conditioned by the 1P indicator.

An entry of U1 through U8 in positions 71 and 72 lets the programmer control the operation of input, output, update, and combined files at run time. If the specified indicator is on at program initialization, the file is opened. If the indicator is not on, the file is not opened and is ignored during processing. The U1 through U8 indicators can be set as follows:

- By the OS/400 control language.
- When used as a resulting indicator for a calculation operation or as field indicators on the input specifications. Setting the U1 through U8 indicators in this manner has no effect on file conditioning.

The UC entry is required for programmer control of only the first file opening. If a file is opened by RPG/400 and later is closed by the CLOSE operation, the programmer can reopen the file (by the OPEN operation) and the UC entry is not required in positions 71 and 72.

Positions 73-74

Positions 73 and 74 must be blank.

Positions 75-80

Positions 75 to 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

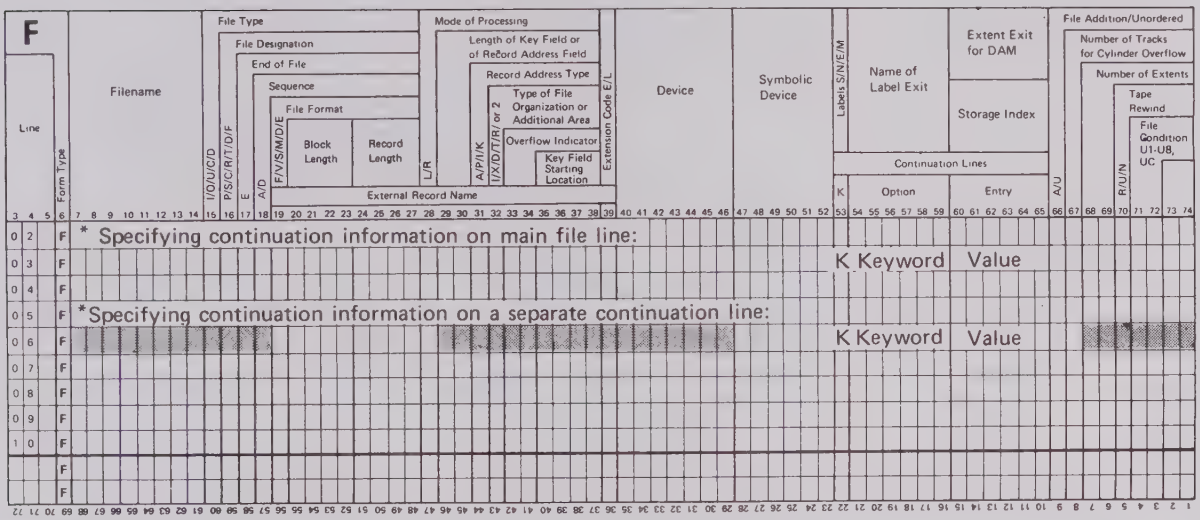
Continuation Line

Continuation lines can be specified on the file description specification to provide additional information about the file being defined. Any number of continuation lines can be specified. A continuation line is indicated by a K in position 53 (see Figure 2-2 on page 2-19).

A continuation line can be specified on the main file description specification line if the functions use positions 54 through 65 for their definition; however, the keywords SFILE, RENAME, IGNORE, and PLIST cannot be defined on the main file description specification line. To specify the continuation line information on the main file description specification line, enter K in position 53 and the valid entries in positions 54 through 65.

Valid entries for continuation lines are given in the two summary charts below.

File Description Specifications



Shaded positions must be blank.

Figure 2-2. RPG Control and File Description Specifications Form – Continuation Line

Continuation Line Summary Chart

Table 2-3 (Page 1 of 2). Continuation Line Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	F	Identification for a file description specification.
7-18		Blank	These positions must be blank for a separate continuation line.
19-28		External name of record format	These positions are used to specify the external name of the record format that is to be renamed (RENAME) or ignored (IGNORE).
29-46		Blank	These positions must be blank if the continuation information is specified on a separate continuation line.
47-52	Record number field for SFILE	Numeric field name	For the SFILE options, these positions must specify the name of a Relative Record Number (RECNO) field. For other continuation line options, these positions must be blank.
53	Continuation line	K	Indicates a continuation line.

Table 2-3 (Page 2 of 2). Continuation Line Summary Chart

Positions	Name	Entry	Explanation
54-59, 60-65			These positions are used together. Positions 54 through 59 specify the option, while positions 60 through 65 provide further explanation of the option. See "Continuation Line Options Summary Chart" on page 2-21.
66-74		Blank	These positions must be blank for a separate continuation line.
75-80		Optional	This space is available for comments.

Continuation Line Options Summary Chart

The valid entries for positions 54 through 65 are:

Table 2-4 (Page 1 of 5). Continuation Line Options

Option (54-59)	Entry (60-65)	Explanation
COMIT	Blank	This file is specified for commitment control. Use the COMIT and ROLBK operation codes to put changes to this file into groups so that the changes all happen together, or they do not happen at all.
ID	Field name	This is the name of a 10-character alphanumeric field which does not need to be further defined. This field contains the name of the program device that supplied the record being processed in the file. The field is updated each time a record is read from a file. Also, you may move a program device name into this field to direct an output or device-specific input operation (other than a READ-by-file-name or an implicit cycle read) to a different device. When moving a literal into the field, blank the field first, and use the MOVEL operation to place the literal left-justified in the field. Initially, the field is blank. A blank field indicates the requester device. If the requester device is not acquired for your file, you must not use a blank field. The ID field is maintained for each call to a program. If you call program B from within program A, the ID field for program A is not affected. Program B uses a separate ID field. When you return to program A, its ID field has the same value as it had before you called program B. If program B needs to know which devices are acquired to program A, program A must pass this information (as a parameter list) when it calls program B. When you specify ID but not NUM, RPG assumes NUM is present with a value of 1. To determine the name of the requester device, you may look in the appropriate area of the file information data structure. Or, you may process one of the input or output operations described above with the ID field blank. After the operation, the ID field has the name of the requester device.

Table 2-4 (Page 2 of 5). Continuation Line Options

Option (54-59)	Entry (60-65)	Explanation
IGNORE	Blank	This option lets you ignore a record format from an externally described file. On the continuation line, positions 19 through 28 specify the external name of the record format to be ignored, and positions 60 through 67 must be blank. The program runs as if the record format did not exist in the file.
IND	Indicator number	Indicators from 01 to the number specified are saved and restored for each device attached to a mixed or multiple device file. Before an input operation, the indicators for the device associated with the previous input or output operation are saved. After the input operation, the indicators for the device associated with this current input operation are restored. Specify a number from 01 through 99, right-justified, in positions 60 through 65. No indicators are saved and restored if IND is not specified or if the option NUM has the entry 1. If you specified the keyword INDARA, the number you specify for IND must be less than any response indicator you use in your DDS. For example, if you specify INDARA and CF01(55) in your DDS, the maximum value for IND is 54. IND must not be used with shared files. When you specify IND but not NUM, RPG assumes NUM is present with a value of 1.
INFDS	Data structure name	This entry lets you obtain exception/error information for a file by defining and naming a data structure to contain the exception/error information. The data structure name is entered in positions 60 through 65. If INFDS is specified for more than one file, each associated data structure must have a unique name.
INFSR	Subroutine name	The file exception/error subroutine named in positions 60 through 65 may receive control following file exception/errors. The subroutine name may be *PSSR, which indicates the user defined program exception/error subroutine is to be given control for errors on this file.

Table 2-4 (Page 3 of 5). Continuation Line Options

Option (54-59)	Entry (60-65)	Explanation
NUM	Maximum number of devices	The number specified must be greater than zero and right-justified in positions 60 through 65. The lesser of this number and the number of devices defined for the WORKSTN file on the create-file command is the maximum number of devices that this file can acquire. With a shared file, RPG/400 does not use the NUM value to restrict the number of acquired devices. When you specify ID, IND, or SAVDS but not NUM, RPG/400 assumes NUM is present with a value of 1.
PASS	*NOIND	Specify PASS *NOIND on the file description specification continuation line for a program described WORKSTN file if you are taking responsibility for passing indicators on input and output. With PASS *NOIND, RPG does not pass indicators to data management on output and does not receive them on input. Pass indicators by describing them as fields (in the form *INxx, *IN, or *IN,xx) in the input or output record. They must be specified in the sequence required by the data description specifications (DDS). You can use the DDS listing to determine this sequence. If you do not specify PASS *NOIND and you use the keyword INDARA in the DDS for the WORKSTN file, RPG/400 does pass indicators to data management on output and receive them from data management on input.
PLIST	Parameter list name	This entry is valid only when the device specified in positions 40 through 46 of the main file description line is SPECIAL. Positions 60 through 65 give the name of the parameter list that is to be passed to the special routine. The parameters identified by this entry are added to the end of the parameter list passed by the RPG/400 program.
PRTCTL	Data structure name	The dynamic printer control option is being used. The data structure specified in positions 60 through 65 refers to the forms control information and line count value. The PRTCTL option is valid only for a program described file.

Table 2-4 (Page 4 of 5). Continuation Line Options

Option (54-59)	Entry (60-65)	Explanation
RECNO	Field name	This entry is optional for disk files to be processed by relative record number. A RECNO field must be specified for output files processed by relative record number, output files that are referenced by a random WRITE calculation operation, or output files that are used with ADD on the output specifications. RECNO can be specified for input/update files. RPG/400 places the relative record number of the record retrieved in the field named in positions 60 through 65 for all operations that reposition the file (such as READ, SETLL, or OPEN). The field name specified in positions 60 through 65 must be defined as numeric with zero decimal positions. The field length must be sufficient to contain the longest record number for the file. RECNO is valid for DISK files only. The contents of positions 60 through 65 may be invalid when the RPG/400 compiler does the blocking and unblocking of records.
RENAME	Record format name	This entry, which is optional, allows you to rename record formats in an externally described file. Positions 19 through 28 of the continuation line specify the external name of the record format that is to be renamed. Positions 60 through 67 specify the name of the record as it is used in the program. The external name is replaced by this name in the program.
SAVDS	Data structure name	This is the name of the data structure that is saved and restored for each device Before an input operation, the data structure for the device associated with the previous input or output operation is saved. After the input operation, the data structure for the device associated with this current input operation is restored. This data structure cannot be a data area data structure, file information data structure, or program status data structure, and it cannot contain a compile-time array or prerun-time array. If SAVDS is not specified, no saving and restoring is done. SAVDS must not be specified for shared files. When you specify SAVDS but not NUM, RPG/400 assumes NUM is present with a value of 1.

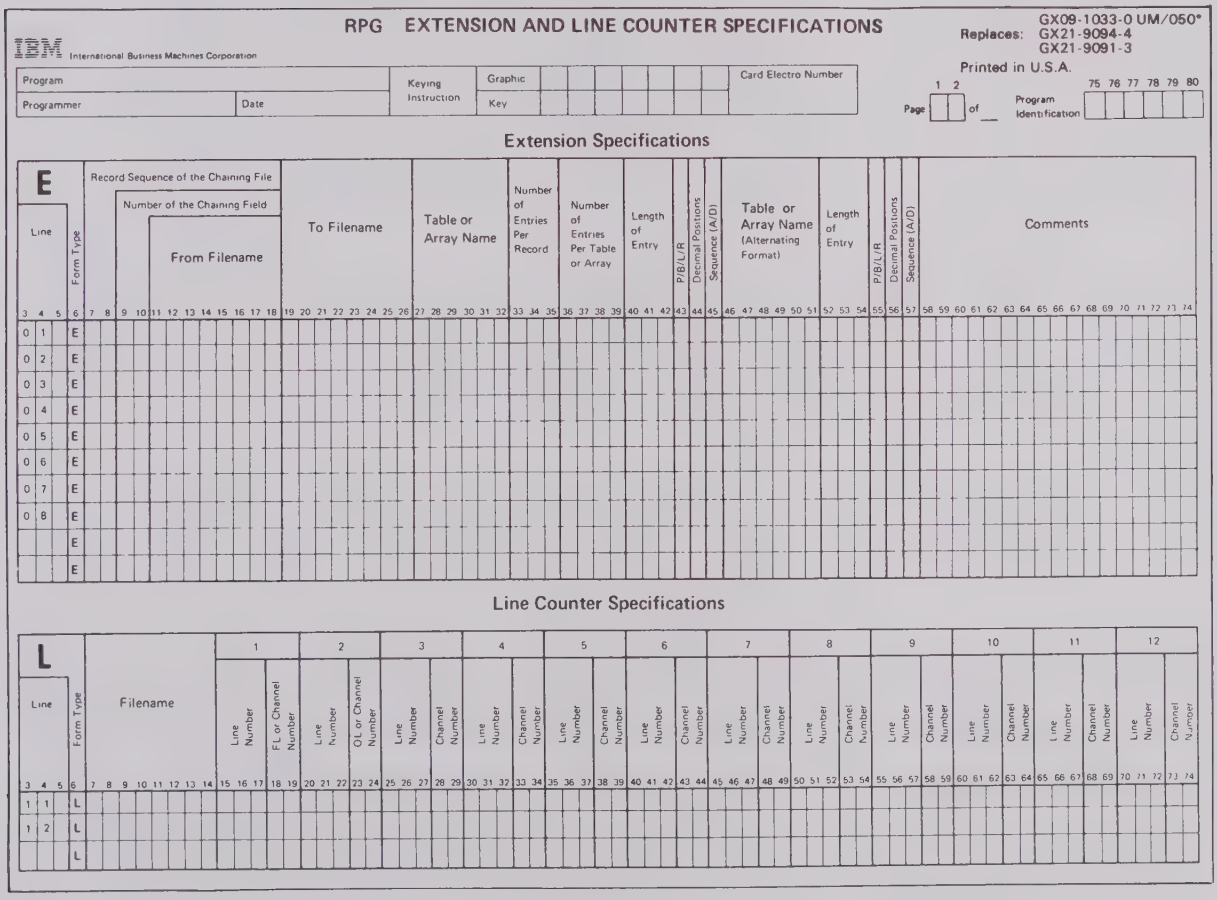
Table 2-4 (Page 5 of 5). Continuation Line Options

Option (54-59)	Entry (60-65)	Explanation
SFILE	Record format name	If the main file description line contains E in position 19 and WORKSTN in positions 40 through 46, this option must be used to define any subfiles to be used in the file. Positions 60 through 67 must specify the RPG/400 name of the record format to be processed as a subfile. Positions 47 through 52 must specify the name of the relative record number field for this subfile. RPG/400 places the relative record number of any record retrieved by a READC or CHAIN operation into the field named in positions 47 through 52. This field is also used to specify the record number that RPG/400 uses for a WRITE operation to the subfile or for output operations that use ADD. The field name specified in positions 47 through 52 must be defined as numeric with zero decimal positions. The field must have enough positions to contain the largest record number for the file (see the SFLSIZ keyword in the *Programming: Data Description Specifications Reference*. Relative record number processing is implicitly defined as part of the SFILE definition. If multiple subfiles are defined, each subfile requires a separate continuation line.
SLN	Field name	This option allows you to specify the name of a start line number (SLN) field. The SLN field determines where a record format will be written to a display file. The main file description line must contain WORKSTN in positions 40 through 46 and a C or O in positions 15. The data description specifications for the file must specify the keyword SLNO(*VAR) for one or more record formats. When you specify SLN on the continuation line, the SLN field will automatically be defined in the program as a numeric field with length of 2 and with 0 decimal positions.

Chapter 3. Extension Specifications

Extension specifications describe all record address files, arrays, and tables. A maximum of 200 arrays and tables can be used in a program. Enter these specifications on the RPG/400 Extension and Line Counter Specifications (see Figure 3-1).

Figure 3-2 on page 3-5 shows the valid positions for arrays, tables, and record address files.


RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

GX09-1033-0 UM/050*
Replaces: GX21-9094-4
GX21-9091-3
Printed in U.S.A.

IBM International Business Machines Corporation

Program | Keying Instruction | Graphic Key | Card Electro Number
Programmer | Date

Page 1 2 of __ | Program Identification 75 76 77 78 79 80

Extension Specifications

E | Line 3 4 5 | Form Type 6 | Record Sequence of the Chaining File 7 8 | Number of the Chaining Field 9 10 | From Filename 11 12 13 14 15 16 17 18 | To Filename 19 20 21 22 23 24 25 26 | Table or Array Name 27 28 29 30 31 32 | Number of Entries Per Record 33 34 35 | Number of Entries Per Table or Array 36 37 38 39 | Length of Entry 40 41 42 | P/B/L/R 43 | Decimal Positions 44 | Sequence (A/D) 45 | Table or Array Name (Alternating Format) 46 47 48 49 50 51 | Length of Entry 52 53 54 | P/B/L/R 55 | Decimal Positions 56 | Sequence (A/D) 57 | Comments 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74

Line Counter Specifications

L | Line 3 4 5 | Form Type 6 | Filename 7 8 9 10 11 12 13 14 | 1: Line Number 15 16 17 | FL or Channel Number 18 19 | 2: Line Number 20 21 22 | OL or Channel Number 23 24 | 3: Line Number 25 26 27 | Channel Number 28 29 | 4: Line Number 30 31 32 | Channel Number 33 34 | 5: Line Number 35 36 37 | Channel Number 38 39 | 6: Line Number 40 41 42 | Channel Number 43 44 | 7: Line Number 45 46 47 | Channel Number 48 49 | 8: Line Number 50 51 52 | Channel Number 53 54 | 9: Line Number 55 56 57 | Channel Number 58 59 | 10: Line Number 60 61 62 | Channel Number 63 64 | 11: Line Number 65 66 67 | Channel Number 68 69 | 12: Line Number 70 71 72 | Channel Number 73 74

Figure 3-1. RPG Extension and Line Counter Specifications Form

Extension Specification Summary Chart

Table 3-1 (Page 1 of 3). Extension Specifications Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page Number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	E	Identification for an extension specification.
7-10		Blank	
11-18	From file name	Blank	The array or table is loaded at compilation time or by input or calculation specifications.
		Record address file name	Name of the record address file.
		Array or table file name	The array or table file loaded at prerun time.
19-26	To file name	Blank	The array or table is not written at the end of the program.
		Name of an input or update file containing data records	File processed with the record address file named in positions 11 through 18.
		Name of an output or combined file	File (output or combined) to which an array or table is to be written.
27-32	Table or array name	Table or array name	The name of array or table used in the program.
33-35	Number of entries per record	Blank	The array is loaded by input or calculation specifications.
		1-999	Number of array or table entries in each array or table input record.
36-39	Number of entries per array or table	1-9999	Maximum number of array or table entries.
40-42	Length of entry	1-256	Length of each element in the array or table named in positions 27 through 32.

Table 3-1 (Page 2 of 3). Extension Specifications Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
43	Packed/binary, sign left/right	Blank	The data for the array or table is in zoned decimal format or is character format.
		P	The data for the array or table is in packed decimal format.
		B	The data for the array or table is in binary format.
		L	The data for a numeric array or table element has a preceding (left) plus or minus sign.
		R	The data for a numeric array or table element has a following (right) plus or minus sign.
44	Decimal positions	Blank	Character array or table.
		0-9	Number of positions to the right of the decimal in numeric array or table elements.
45	Sequence	Blank	No particular sequence.
		A	Ascending sequence.
		D	Descending sequence.
46-51	Table or array name (alternating format)	Table or array name (alternating format)	The name of the table or array used in the program.
52-54	Length of entry	1-256	Length of each element in the array or table named in positions 46-51.
55	Packed/binary, sign left/right	Blank	The data for the array or table is in zoned decimal format or is character format.
		P	The data for the array or table is in packed decimal format.
		B	The data for the array or table is in binary format.
		L	The data for a numeric array or table element has a preceding (left) plus or minus sign.
		R	The data for a numeric array or table element has a following (right) plus or minus sign.

Table 3-1 (Page 3 of 3). Extension Specifications Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
56	Decimal positions	Blank	Character array or table.
		0-9	Number of positions to the right of the decimal in numeric array or table elements.
57	Sequence	Blank	No particular sequence.
		A	Ascending sequence.
		D	Descending sequence.
58-74		Comments	Used to document the purpose of each specification line.
75-80		Optional	This space is available for comments.

Extension Specification Statement

Position 6 (Form Type)

An E must appear in position 6 to identify this line as an extension specifications statement.

Positions 7-10

Positions 7 through 10 must be blank.

Positions 11-18 (From File Name)

Entry	Explanation
Blank	The array or table is loaded at compilation time, or the array is loaded by input or calculation specifications.
Record address file name	Name of the record address file.
Array or table file name	Name of the array or table file loaded at prerun time.

Use positions 11 through 18 to name an array file, table file, or record address file. File names must begin in position 11. The record address file name must always be entered in these positions. The file name of every prerun-time array or table used in the program must be entered in these positions. Leave positions 11 through 18 blank for compile-time arrays or tables and for run-time arrays loaded with input and/or calculation specifications.

Table 3-2 on page 3-6 shows the relationship between positions 11 through 18 and positions 19 through 26.

When an array or table is loaded at compilation time, it is compiled along with the source program and included in the program. Such an array or table does not need to be loaded separately every time the program is run. Only those arrays and tables that contain constant data should be compiled with the program.

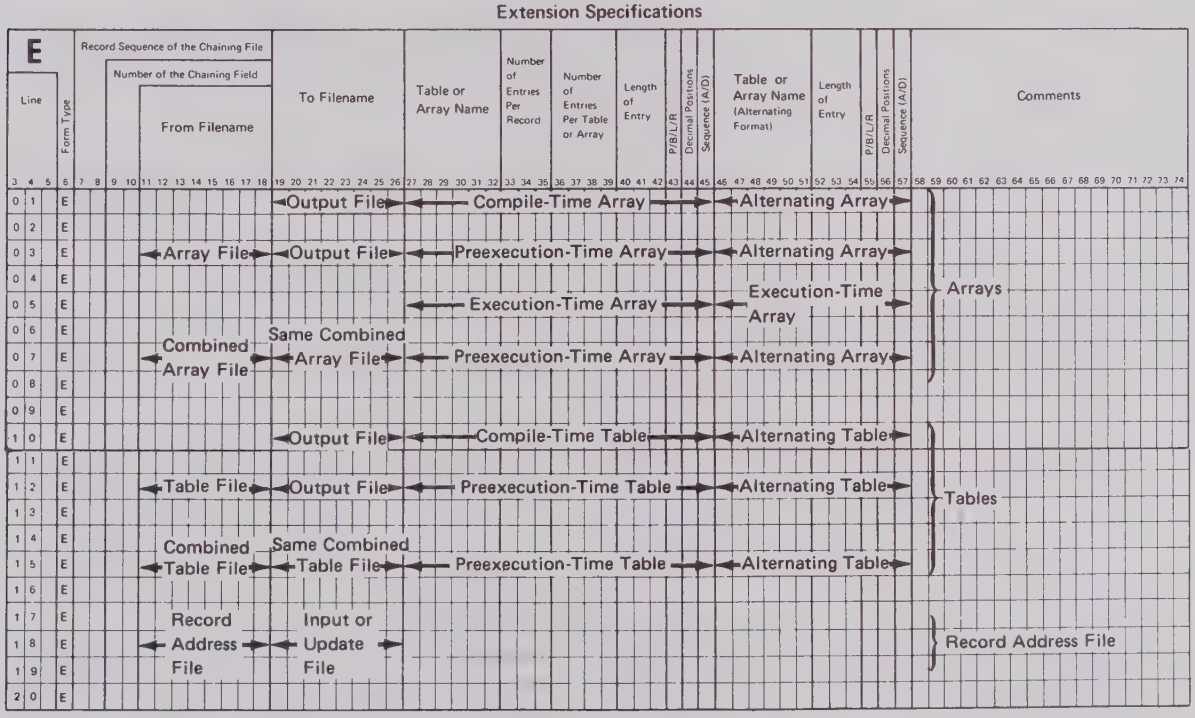


Figure 3-2. Possible Extension Specifications Entries

Positions 19-26 (To File Name)

Entry	Explanation
Blank	The array or table is not written at end of program.
Name of an input or update file containing data records	File processed with the record address file named in positions 11 through 18.
Name of an output or combined file	Output file to which an array or table is to be written, or the same file name (must be a combined table file) specified in positions 11 through 18 if the output array or table is to replace input in the same file. The file should be externally described as a physical file.

If a record address file is named in positions 11 through 18, the name of the input or update file that contains the data records to be processed must be entered in positions 19 through 26.

If an array or table is to be written, enter the file name of the output or combined file in positions 19 through 26. This file must also be named in the file description specifications. An array or table can be written to only one output device. Leave positions 19 through 26 blank if the array or table is not to be written.

If an array or table is assigned to an output file, it is automatically written at the end of the run if LR is on. The array or table is written after all other records are written in the format used when it was entered.

If an array or table is to be written to the same file from which it was read, the same file name must be entered in positions 11 through 18 and in positions 19 through 26. This file must be specified as a combined file (C in position 15) in the file description specifications.

Table 3-2. From and To File Name Entries

Type of File	From File Name (Positions 11-18)	To File Name (Positions 19-26)
Array or table files loaded at prerun time	If an array or table loaded at prerun time is being defined (positions 27 through 57), enter the name of the file that contains the array or table.[1]	If the array or table being defined is being written out after it is updated, enter the name of the output file or the combined array file if it is to be written to the same file that was assigned to it in the file description specifications.[1] If the array or table is not being written out, leave these positions blank.
Arrays or tables loaded at compile time	This entry must be blank.	Enter the name of the output file if the array or table is to be written out at the end of the program.[1]
Arrays loaded by input or calculation specifications	This entry must be blank.	This entry must be blank.
Record address file	Enter the name of the record address file.[1]	Enter the name of the file that contains the data records to be processed by the record address file.[1]

[1] These entries must be left-adjusted.

Positions 27-32 (Array or Table Name)

Entry	Explanation
Array or table name	The name of the array or table used in the program.

Use positions 27 through 32 to name the array or table.

Positions 33-35 (Number of Entries per Record)

Entry	Explanation
Blank	The array is loaded by input or calculation specifications.
1-999	Number of array or table entries in each array or table input record.

Use positions 33 through 35 to indicate the exact number of array or table entries in each array or table input record. The number must end in position 35. Every array or table input record except the last must contain the number of entries indicated in positions 33 through 35. The last record can contain fewer entries than indicated, but not more. Comments can be entered on table input records in the positions following the table entries.

If two arrays or tables are in alternating format in one file, each array or table input record must contain the corresponding entries from each array or table. The corresponding entries from the two arrays or tables are considered one entry and must be on the same record. For more information on alternating arrays or tables, see

If positions 27 through 32 contain an array name, the following rules apply to the use of positions 11 through 18 and 33 through 35:

- For a prerun-time array, positions 11 through 18 must contain a file name and positions 33 through 35 must have an entry.
- For a compile-time array, positions 11 through 18 must be blank and positions 33 through 35 must have an entry.
- For an run-time array, positions 11 through 18 and positions 33 through 35 must be blank.

Positions 36-39 (Number of Entries per Array or Table)

Entry	Explanation
1-9999	Maximum number of array or table entries.

Use positions 36 through 39 to indicate the maximum number of entries that can be contained in the array or table named in positions 27 through 32. This number applies to one array or table or to two arrays or tables in alternating format. The number entered must end in position 39.

Because the number of entries for two arrays or tables written in alternating format must be the same, the number in these positions also gives the number of entries in the second array or table specified in positions 46 through 51.

Positions 40-42 (Length of Entry)

Entry	Explanation
1-256	Length of each element in the array or table named in positions 27 through 32.

If L or R is specified in positions 43 or 55, the length includes the sign position.

If two arrays or tables are entered in alternating format, the specification in positions 40 through 42 applies to the array or table whose entry appears first in the record.

Position 43 (Packed/Binary, Sign Left/Right)

Entry	Explanation
Blank	The data for the array or table (1) is in zoned decimal format, or (2) is character data, or (3) is loaded through input or calculation specifications.
P	The data for the array or table is in packed decimal format.
B	The data for the array or table is in binary format.
L	The data for a numeric array or table element has a preceding (left) plus or minus sign.
R	The data for a numeric array or table element has a following (right) plus or minus sign.

The entry in position 43 specifies the format of the data in the records in the file. This entry has no effect on the format used for internal processing of the array or table in the program.

Position 44 (Decimal Positions)

Entry	Explanation
Blank	Character array or table
0-9	Number of positions to the right of the decimal in numeric array or table elements.

Use position 44 to indicate the number of decimal positions in a numeric array or table element. Position 44 must always have an entry for a numeric array or table. If the entries in an array or table have no decimal positions, enter a 0.

If two arrays or tables are entered in alternating format, the specification in this position applies to the array or table containing the entry that appears first on the record.

Position 45 (Sequence)

Entry	Explanation
Blank	No particular sequence
A	Ascending sequence
D	Descending sequence.

Use position 45 to describe the sequence (either ascending or descending) of the data in an array or table loaded at prerun time or compile time.

When an entry is made in position 45, the array or table is checked for the specified sequence at the time the array or table is loaded with data. If a prerun-time array or table is out of sequence, control passes to the RPG exception/error handling routine.

Ascending sequence means that the array or table entries start with the lowest data entry (according to the collating sequence) and go to the highest. Descending sequence means that the array or table entries start with the highest data entry and go to the lowest.

If two arrays or tables are entered in alternating format, the entry in position 45 applies to the array or table containing the entry that appears first on the record.

When the LOKUP operation is used to search an array or table for an entry to determine whether the entry is high or low compared with the search argument, a sequence must have been specified (A or D) for the array or table. See "LOKUP (Lookup)" on page 9-82 for more information.

An run-time array (loaded by input and/or calculation specifications) is not sequence checked. However, an A or D entry must be specified if a high or low LOKUP operation is processed. Sequence must be specified if the SORTA operation code is used with the array.

Positions 46-57

The fields in positions 46 through 57 have the same significance and require the same type of entries as the fields with corresponding titles in positions 27 through 32 and 40 through 45.

Positions 46 through 57 can be used to describe a second array:

- For compile-time and prerun-time arrays, the array described in positions 46 through 57 is loaded in alternating format with the array named in positions 27 through 32.
- For run-time arrays, positions 46 through 57 can be used to describe a second run-time array that is loaded independently of the array named in positions 27 through 32.

See the previous discussion on positions 27 through 45 for information about correct specifications. Leave positions 46 through 57 blank for a single array or table.

Positions 58-74 (Comments)

Positions 58 through 74 may be used for comments.

Positions 75-80

Positions 75 through 80 can be used for comments, or left blank.

These positions are not printed contiguously with positions 6-74 on the compiler listing.

Chapter 4. Line Counter Specifications

Line counter specifications can be used for each program described printer file to indicate the length of the form and the number of lines to print on a page. These entries are specified on the RPG Extension and Line Counter Specifications (see Figure 4-1). Line counter specifications may be used for each printer file in your program. If line counter specifications are used, position 39 of the file description specifications for the PRINTER device must contain an L.

A form length and an overflow line specified by OS/400 override commands override any program specifications. If no override commands are used for the printer file, the program specification of form length and overflow line is used. If there are no overrides and no program specifications, the form length and the overflow line specified in the device file are used.

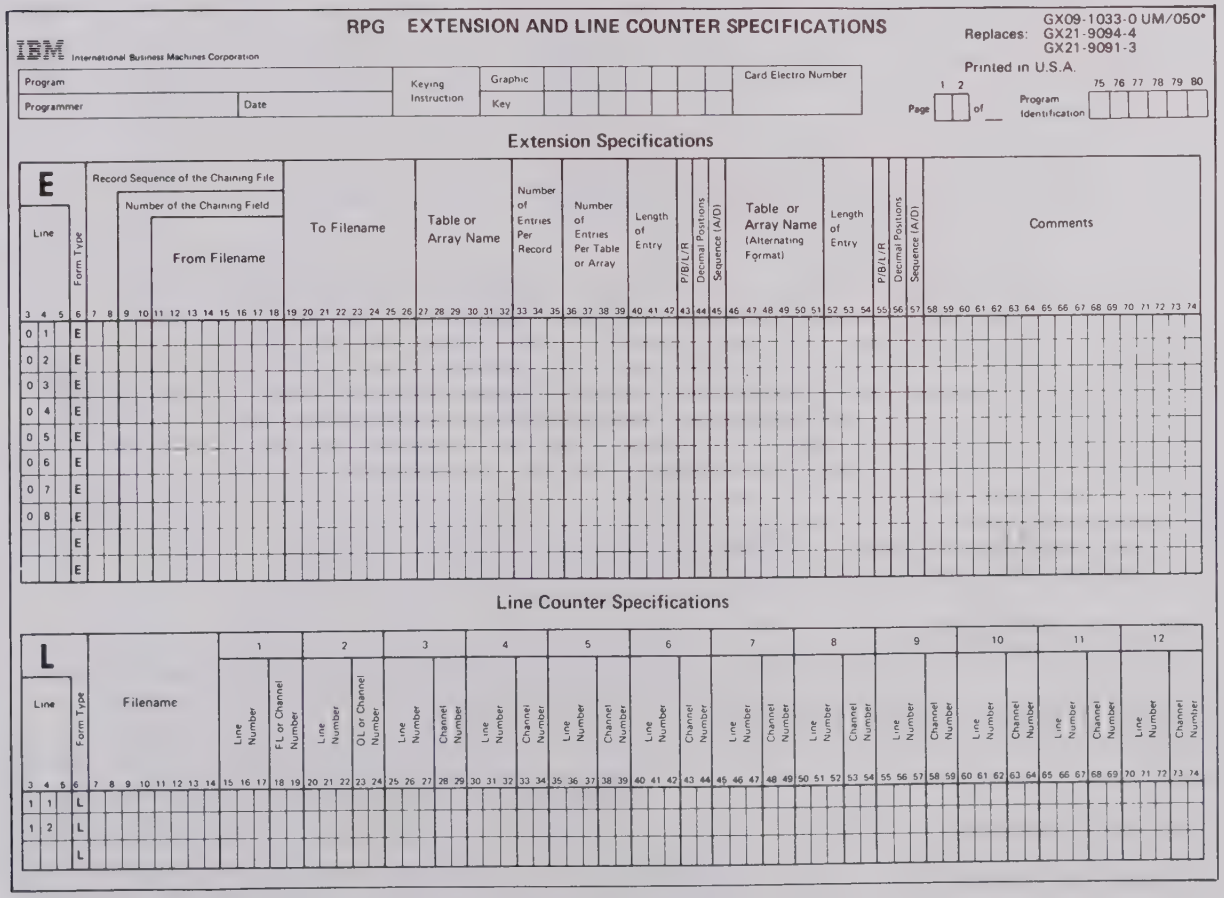


Figure 4-1. RPG Extension and Line Counter Specifications Form

Line Counter Specification Summary Chart

Table 4-1. Line Counter Specification Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	L	Identification for a line counter specification.
7-14	File name	A valid file name	File name of the printer file as previously specified on file description specification form.
15-17	Line number—number of lines per page	2-112	The number of printing lines available is 2 through 112.
18-19	Form length	FL	Indicates that the entry in positions 15 through 17 is the form length. Positions 18 and 19 must contain FL if positions 15 through 17 contain an entry.
20-22	Overflow line number	2-112	The line number specified is the overflow line.
23-24	Overflow line	OL	Indicates that the preceding entry is the overflow line. Positions 23 and 24 must contain OL if positions 20 through 22 contain an entry.
25-74		Blank	
75-80		Optional	This space is available for comments.

Line Counter Specification Statement

Position 6 (Form Type)

An L must be entered in position 6 to identify this line as a line counter specifications statement.

Positions 7-14 (File Name)

Entry	Explanation
A valid file name	File name of the program described printer file as previously defined on the file description specifications. The file name must begin in position 7.

Positions 15-17 (Number of Lines Per Page)

Entry	Explanation
2-112	The number of printing lines available is 2 through 112.

Use positions 15 through 17 to specify the exact number of lines available on the form or page to be used. The entry must end in position 17. Leading zeros can be omitted.

Positions 18-19 (Form Length)

Entry	Explanation
FL	Form length.

Use positions 18 and 19 to indicate that the preceding entry (positions 15 through 17) is the form length. Positions 18 and 19 must contain the entry FL if positions 15 through 17 contain an entry.

Changing the form length does not require recompiling the program. The override to the compiled value can be specified by an OS/400 override command.

Positions 20-22 (Overflow Line Number)

Entry	Explanation
2-112	The line number specified is the overflow line.

Use positions 20 through 22 to specify the overflow line number. The overflow line number must be less than or equal to the form length. The entry must end in position 22. Leading zeros can be omitted. When the line that is specified as the overflow line is printed, the overflow indicator turns on. Changing the overflow line does not require recompiling the program. The override to the compiled value can be specified by an OS/400 override command.

Positions 23-24 (Overflow Line)

Entry	Explanation
OL	Overflow line.

Use positions 23 and 24 to indicate that the preceding entry (positions 20 through 22) is the overflow line number. Positions 23 and 24 must contain OL if positions 20 through 22 contain an entry.

Positions 25-74

Positions 25 through 74 must be blank.

Positions 75-80

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Chapter 5. Input Specifications

For a program described input file, input specifications describe the types of records within the file, the sequence of the types of records, the fields within a record, the data within the field, indicators based on the contents of the fields, control fields, fields used for matching records, and fields used for sequence checking. For an externally described file, input specifications are optional and can be used to add RPG/400 functions to the external description. Input specifications are also used to describe data structures. The RPG/400 Input Specification is shown in Figure 5-1 on page 5-2.

As Figure 5-1 on page 5-2 shows, for program described files, entries on input specifications are divided into the following categories:

- Record identification entries (positions 7 through 42), which describe the input record and its relationship to other records in the file.
- Field description entries (positions 43 through 70), which describe the fields in the records. Each field is described on a separate line, below its corresponding record identification entry.

For externally described files, entries on input specifications are divided into the following categories:

- Record identification entries (positions 7 through 14, and 18 and 19), which identify the record (the externally described record format) to which RPG/400 functions are to be added.
- Field description entries (positions 21 through 30, 53 through 62, and 65 through 70), which describe the RPG/400 functions to be added to the fields in the record. Field description entries are written on the lines following the corresponding record identification entries.

For data structures, entries on input specifications are divided into the following categories:

- Data structure statements (positions 7 through 12, 17 through 30, and 44 through 51), which define data structures.
- Data structure subfield specifications (positions 21 through 30, and 43 through 58), which describe the subfields of the data structures. Data structure subfield specifications are written on the lines following the data structure statements.

This chapter is organized in the following sequence:

- Input specifications summary charts
- Entries for program described files
- Entries for externally described files
- Entries for data structures.

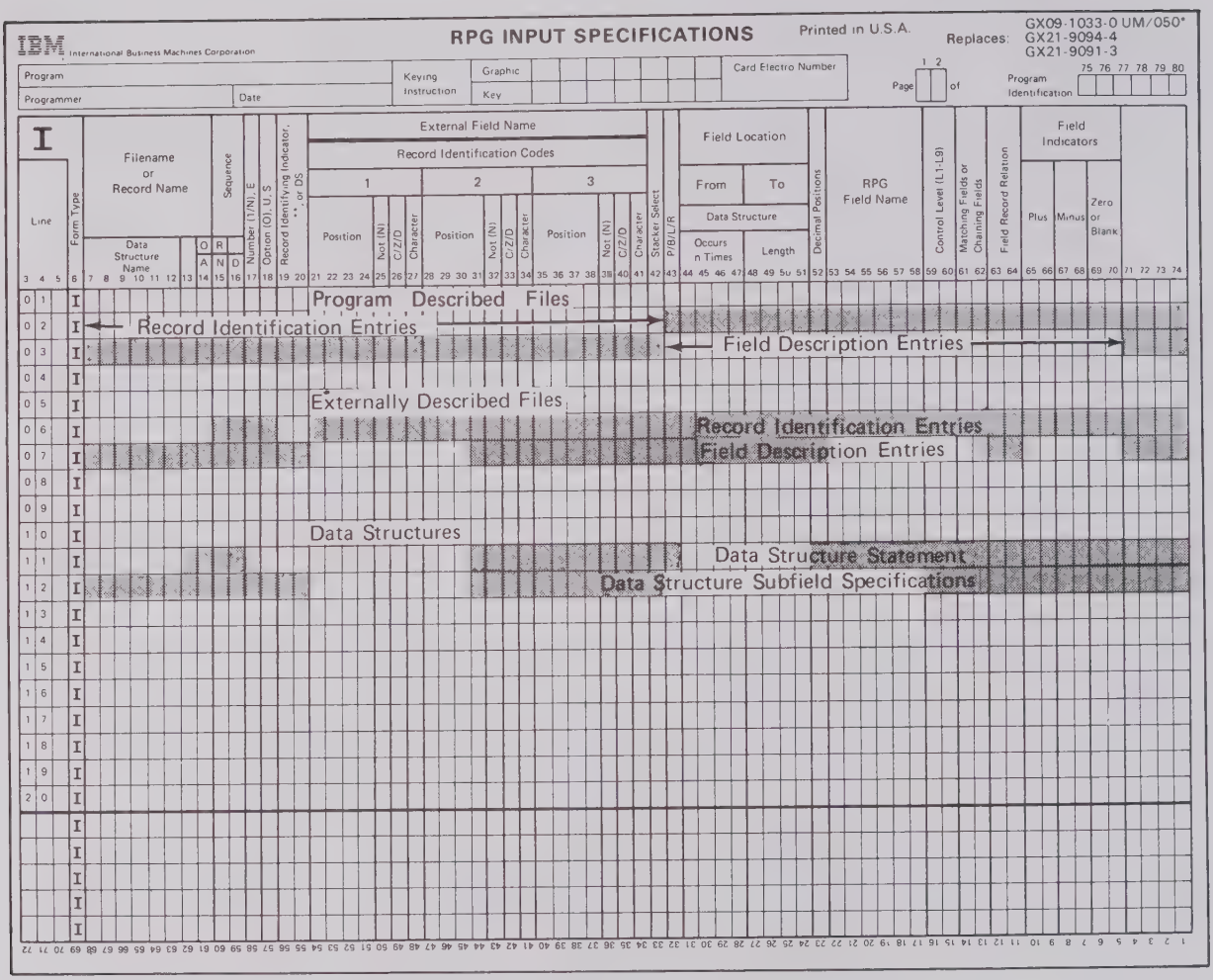


Figure 5-1. RPG/400 Input Specifications Form

Input Specifications Summary Charts

Program Described Files, Record Identification Entries

Table 5-1 (Page 1 of 2). Program Described Files, Record Identification Entries Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	I	Identification for an input specification.
7-14	File name	A valid file name	Must be the same file name that appears on the file description specifications for the input file.
14-16	AND/OR	AND or OR	Enter AND in positions 14 through 16 of the next specification line if more than three record identification codes are needed to identify the record. Enter OR in positions 14 and 15 if two or more record types have common fields.
15-16	Sequence	Any two alphabetic characters	The program does not check for special sequence.
		Any two-digit number	The program checks for special sequence within the group.
17	Number	Blank	The program does not check record types for a special sequence (positions 15 and 16 have alphabetic entries).
		1	Only one record of this type can be present in the sequenced group.
		N	One or more records of this type can be present in the sequenced group.
18	Option	Blank	The record type must be present if sequence checking is specified.
		O	The record type is optional (it may or may not be present) if sequence checking is specified.

Table 5-1 (Page 2 of 2). Program Described Files, Record Identification Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
19-20	Record identifying indicator	01-99	General indicator.
		L1-L9, or LR	Control level indicator used for a record identifying indicator.
		H1-H9	Halt indicator.
		U1-U8	External indicator.
		RT	Return indicator.
		**	Look-ahead field (not an indicator). Look-ahead can only be used with a primary or secondary file.

Use Positions 21 through 41 to enter the record identification codes explained below:

Table 5-2. Program Described Files, Record Identification Codes Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
21-24, 28-31, 35-38	Position	Blank	No record identification code is present.
		1-9999	The position in the record that contains the record identification code.
25, 32, 39	Not	Blank	Record identification code must be present.
		N	Record identification code must not be present.
26, 33, 40	C/Z/D	C	Entire character.
		Z	Zone portion of character.
		D	Digit portion of character.
27, 34, 41	Character	Any character	Enter the identifying character that is to be compared to the character in the position specified in the input record.
42-74		Blank	
75-80		Optional	This space is available for comments.

Program Described Files, Field Description Entries

Table 5-3 (Page 1 of 2). Program Described Files, Field Description Entries Summary Chart

Positions	Name	Entry	Explanation
7-42		Blank	
43	Packed/binary, sign left/right	Blank	The input field is in zoned decimal format or character format.
		P	The input field is in packed decimal format.
		B	The input field is in binary format.
		L	The numeric input field has a preceding (left) plus or minus sign.
		R	The numeric input field has a following (right) plus or minus sign.
44-47	From	1-9999	Specifies the beginning position of the field in the record.
48-51	To	1-9999	Specifies the end position of the field in the record.
52	Decimal positions	Blank	Character field.
		0-9	Number of decimal positions in numeric field.
53-58	Field name	Symbolic name	Field name, data structure name, subfield name, array name, array element, `PAGE`, `PAGE1-PAGE7`, `*IN`, or `*INxx`.
59-60	Control level	Blank	This is not a control field. Control level indicators cannot be used with full procedural files.
		`L1-L9`	This field is a control field.
61-62	Match fields	Blank	This is not a match field.
		`M1-M9`	This field is a match field. Match fields are valid only for primary and secondary files.

Table 5-3 (Page 2 of 2). Program Described Files, Field Description Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
63-64	Field record relation	Blank	The field is common to all record types.
		01-99	General indicators.
		L1-L9	Control level indicators.
		MR	Matching record indicator.
		U1-U8	External indicators.
		H1-H9	Halt indicators.
		RT	Return indicator.
65-70	Field indicators	Blank	No indicator specified.
		01-99	Field indicator.
		H1-H9	Halt indicator.
		U1-U8	External indicators.
		RT	Return indicator.
71-74		Blank	
75-80		Optional	This space is available for comments.

Externally Described Files, Record Identification Entries

Table 5-4 (Page 1 of 2). Externally Described Files, Record Identification Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	I	Identification for an input specification.
7-14	Record name	Record format name	The RPG/400 name of the record format. A file name cannot be used.
15-18	Sequence	Blank	These positions must be blank.

Table 5-4 (Page 2 of 2). Externally Described Files, Record Identification Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
19-20	Record identifying indicators	Blank	No record identifying indicator.
		01-99	General indicator.
		L1-L9, LR	Control level indicator used for record identifying indicator.
		H1-H9	Halt indicator.
		U1-U8	External indicator.
		RT	Return indicator.
21-41	Record identification code	Blank	Record format names are used to determine the record types used in the program.
42-74		Blank	
75-80		Optional	This space is available for comments.

Externally Described Files, Field Description Entries

Table 5-5 (Page 1 of 2). Externally Described Files, Field Description Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
7-20		Blank	
21-30	Rename	Field name	If a field within a record in an externally described field is to be renamed, enter the external name of the field in these positions.
31-52		Blank	
53-58	RPG/400 field name	Field name	The name of the field as it appears in the external record description (if 6 characters or less) or the field name that replaces the externally defined field name in positions 21 through 30.
59-60	Control level	Blank	Field is not a control field.
		L1-L9	This field is a control field.
61-62	Match fields	Blank	Field is not a match field.
		M1-M9	The field is a match field.
63-64		Blank	

Table 5-5 (Page 2 of 2). Externally Described Files, Field Description Entries Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
65-70	Field indicators	Blank	No indicator specified.
		01-99	General indicators.
		H1-H9	Halt indicators.
		U1-U8	External indicators.
		RT	Return indicator.
71-74		Blank	
75-80		Optional	This space is available for comments.

Data Structure Statement Specifications

Table 5-6 (Page 1 of 2). Data Structure Statement Specifications Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	I	Identification for an input specification.
7-12	Data structure name	Blank	The name of the data structure being defined may be omitted.
		Valid data structure name	Enter the name of the data structure being defined.
13-16		Blank	
17	Number	Blank	Subfield definitions for this data structure follow this specification.
		E	Subfield definitions are defined externally. The entry in positions 7 through 12 references the external definition.

Table 5-6 (Page 2 of 2). Data Structure Statement Specifications Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
18	Option	Blank	Not a program status or data area data structure.
		S	A program status data structure. Only one data structure may be designated as the program status data structure.
		U	A data area data structure. RPG/400 retrieves the external data area (named in positions 7 through 12) at initialization and rewrites it at the end of the job. If you put blanks in positions 7 through 12, RPG/400 uses the local data area.
19-20	Record identifying indicator	DS	Indicates a data structure.
21-30	Rename	External name of data structure description	If an externally described data structure is to be renamed, enter the external name of the description in these positions and the name to be used in the program in positions 7 through 12.
31-43		Blank	
44-47	Number of occurrences	Blank	This is not a multiple occurrence data structure.
		1-9999	Entry indicates the number of occurrences of this multiple occurrence data structure.
48-51	Data structure length	Blank	Length of the data structure is either the length specified on the input field specifications if the data structure is an input field or the highest To position specified for a subfield within the data structure if the data structure is not an input field.
		1-9999	Length of the data structure.
52-74		Blank	
75-80		Optional	This space is available for comments.

Data Structure Subfield Specifications

Table 5-7 (Page 1 of 2). Data Structure Subfield Specifications Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
7-20		Blank	
21-30	Rename	Subfield name	If a subfield in an externally described data structure is to be renamed, enter the external name of the subfield in these positions.

Table 5-7 (Page 2 of 2). Data Structure Subfield Specifications Summary Chart

Positions	Name	Entry	Explanation
31-42		Blank	
43	Packed/binary	Blank	Subfield is in zoned decimal format or character format.
		P	Subfield is in packed decimal format.
		B	Subfield is in binary format.
44-47	From	1-9999	Specifies subfield's beginning position in data structure.
48-51	To	1-9999	Specifies subfield's end position in data structure.
44-51	Keywords	Valid keyword	Special keywords define the location of subfields in the program status data structure or the file information data structure. Keywords for the program status data structure are *STATUS, *PROGRAM, *PARMS, and *ROUTINE. Keywords for the file information data structure are *FILE, *RECORD, *OPCODE, *STATUS, and *ROUTINE.
52	Decimal position	Blank	Character subfield.
		0-9	Number of decimal positions in a numeric subfield.
53-58	Subfield name	Valid subfield name	The subfield name or the external name of the subfield or the subfield name that replaces the external subfield name specified in positions 21 through 30.
59-74		Blank	
75-80		Optional	This space is available for comments.

Program Described Files

Position 6 (Form Type)

An I must appear in position 6 to identify this line as an input specification statement.

Record Identification Entries

Record identification entries (positions 7 through 42) for a program described file describe the input record and its relationship to other records in the file.

Positions 7-14 (File Name)

Entry	Explanation
A valid file name	Same file name that appears on the file description specifications for the input file.

Enter the name of the file to be described in these positions. This name must be the same name defined for the file on the file description specifications. This file must be an input file, an update file, or a combined file. The file name must be entered on the first record identification line for each file and can be entered on subsequent record identification lines for that file. All entries describing one input file must appear together; they cannot be mixed with entries for other files.

Positions 15-16 (Sequence)

Entry	Explanation
Any two alphabetic characters	The program does not check for special sequence.
Any two-digit number	The program checks for special sequence within the group.

The numeric sequence entry combined with the number (position 17) and option (position 18) entries causes the program to check the sequence of input records within a file. If the sequence is not correct, control passes to the RPG/400 exception/error handling routine. If AND or OR lines are specified, the sequence entry is made on the main record line of the group, not on the AND or OR lines.

Alphabetic and numeric entries can be made for different records (different record identification lines) in the same file, but records with alphabetic entries must be specified before records with numeric entries.

Alphabetic Entries

Enter any two alphabetic characters in these positions when no sequence checking is to be done. It is common programming practice to specify these codes in a sequence that aids in program documentation. However, it is not necessary to use unique alphabetic entries.

Numeric Entries

Enter a unique numeric code in positions 15 and 16 if one record type must be read before another record type in a file. Numeric entries must be in ascending order, starting with 01, but need not be consecutive. When a numeric entry is used, the appropriate entries must be made in positions 17 and 18.

To specify sequence checking, each record type must have a record identification code, and the record types must be numbered in the order in which they should appear. RPG/400 checks this order as the records are read. If a record type is out of sequence, control passes to the RPG/400 exception/error handling routine.

Sequence numbers ensure only that all records of each record type precede the records of senior sequence numbered record types. The sequence numbers do not ensure that records within a record type are in any certain order. Sequence numbers are unrelated to control levels and do not provide for checking data in fields of a record for a special sequence. Use positions 61 and 62 (matching fields) to indicate that data in fields of a record should be checked for a special sequence.

Position 17 (Number)

Entry	Explanation
Blank	The program does not check record types for a special sequence (positions 15 and 16 have alphabetic entries).
1	Only one record of this type can be present in the sequenced group.
N	One or more records of this type can be present in the sequenced group.

This entry must be used when a numeric entry is made in positions 15 and 16. If an alphabetic entry is made in positions 15 and 16, this entry must be blank.

Position 18 (Option)

Entry	Explanation
Blank	The record type must be present if sequence checking is specified.
O	The record type is optional (that is, it may or may not be present) if sequence checking is specified.

This entry must be blank if positions 15 and 16 contain an alphabetic entry.

Sequence checking of record types has no meaning under the following conditions:

- All record types within a file are specified as optional (alphabetic entry in positions 15 and 16 or 0 entry in position 18).
- All record types within a file are specified as optional except one record type, which has a numeric entry in positions 15 and 16 and a blank entry in position 18.

Positions 19-20 (Record Identifying Indicator, or **)

Entry	Explanation
01-99	General indicator.
L1-L9 or LR	Control level indicator used for a record identifying indicator.
H1-H9	Halt indicator.
U1-U8	External indicator.
RT	Return indicator.

Entry	Explanation
**	Look-ahead field (not an indicator). Look-ahead can be used only with a primary or secondary file.

The indicators specified in these positions are used in conjunction with the record identification codes (positions 21 through 41).

Indicators

Positions 19 and 20 associate an indicator with the record type defined on this line. The normal entry is one of the indicators 01 to 99; however, the control level indicators L1 through L9 and LR can be used to cause certain total steps to be processed. If a control level indicator is specified, lower control level indicators are not set on. The halt indicators H1 through H9 can be used to stop processing. The return indicator (RT) causes return to the calling program.

When a record is selected for processing and satisfies the conditions indicated by the record identification codes, RPG/400 sets the appropriate record identifying indicator on. This indicator can be used to condition calculation and output operations. Record identifying indicators can be set on or set off by the programmer. However, at the end of the cycle, RPG/400 sets all record identifying indicators off before selecting another record.

Look-Ahead Fields

The entry of ** is used for the look-ahead function. This function lets you look at information in the next record in a file. You can look not only at the file currently selected for processing but also at other files present but not selected during this cycle.

Positions 15 and 16 must contain an alphabetic entry. The look-ahead fields are defined in positions 53 through 58 of the lines following the line containing ** in positions 19 and 20. Positions 59 through 74 must be blank.

Any or all of the fields in a record can be defined as look-ahead fields. This definition applies to all records in the file, regardless of their type. If a field is used both as a look-ahead field and as a normal input field, it must be defined twice with different names.

The look-ahead function can be specified only for primary and secondary files and can be specified only once for a file. It cannot be used for full procedural files (identified by an F in position 16 of the file description specifications), or with AND or OR lines.

When a record is being processed from a combined file or an update file, the data in the look-ahead field is the same as the data in the record being processed, not the data in the next record.

If an array element is specified as a look-ahead field, the entire array is classified as a look-ahead field.

Look-ahead fields are filled with nines when all records in the file have been processed so that the end of the file can be recognized.

Positions 21-41 (Record Identification Codes)

Entries in positions 21 through 41 identify each record type in the input file. One to three identification codes can be entered on each specification line. More than three record identification codes can be specified on additional lines with the AND/OR relationship. If the file contains only one record type, the identification codes can be left blank; however, a record identifying indicator entry (positions 19 and 20) and a sequence entry (positions 15 and 16) must be made.

Three sets of entries can be made in positions 21 through 41: 21 through 27, 28 through 34, and 35 through 41. Each set is divided into four groups: position, not, C/Z/D, and character.

The following table shows which categories use which positions in each set.

Category	21-27	28-34	35-41
Position	21-24	28-31	35-38
Not	25	32	39
C/Z/D	26	33	40
Character	27	34	41

Entries in these sets need not be in sequence. For example, an entry can be made in positions 28 through 34 without requiring an entry in positions 21 through 27. Entries for record identification codes are not necessary if input records within a file are of the same type. An input specification containing no record identification code defines the last record type for the file, thus allowing the handling of any record types that are undefined. If no record identification codes are satisfied, control passes to the RPG/400 exception/error handling routine.

Positions 21-24, 28-31, and 35-38 (Position)

Entry	Explanation
Blank	No record identification code is present.
1-9999	The position that contains the record identification code in the record.

In these positions enter the position that contains the record identification code in each record. The position containing the code must be within the record length specified for the file. This entry must be right-adjusted, but leading zeros can be omitted.

Positions 25, 32, and 39 (Not)

Entry	Explanation
Blank	Record identification code must be present.
N	Record identification code must not be present.

Enter an N in this position if the code described must not be present in the specified record position.

Positions 26, 33, and 40 (C/Z/D)

Entry	Explanation
C	Entire character
Z	Zone portion of character
D	Digit portion of character.

This entry specifies what part of the character in the record identification code is to be tested.

Character (C): The C entry indicates that the complete structure (zone and digit) of the character is to be tested.

Zone (Z): The Z entry indicates that the zone portion of the character is to be tested. The zone entry causes the four high-order bits of the character entry (position 27) to be compared with the zone portion of the character in the record position specified in the position entry (positions 21 through 24). The following three special cases are exceptions:

- The hexadecimal representation of an & (ampersand) is 50. However, when an ampersand is coded in the character entry, it is treated as if its hexadecimal representation were C0, that is, as if it had the same zone as A through I. An ampersand in the input data satisfies two zone checks: one for a hexadecimal 5 zone, the other for a hexadecimal C zone.
- The hexadecimal representation of a - (minus sign) is 60. However, when a minus sign is coded in the character entry, it is treated as if its hexadecimal representation were D0, that is, as if it had the same zone as J through R. A minus sign in the input data satisfies two zone checks: one for a hexadecimal 6 zone, the other for a hexadecimal D zone.
- The hexadecimal representation of a blank is 40. However, when a blank is coded in the character entry, it is treated as if its hexadecimal representation were F0, that is, as if it had the same zone as 0 through 9. A blank in the input data satisfies two zone checks: one for a hexadecimal 4 zone, the other for a hexadecimal F zone.

Digit (D): The D entry indicates that the digit portion of the character is to be tested. The four low-order bits of the character are compared with the character specified by the position entry.

Positions 27, 34, and 41 (Character)

In this position enter the identifying character that is to be compared with the character in the position specified in the input record.

The check for record type always starts with the first record type specified. If data in a record satisfies more than one set of record identification codes, the first record type satisfied determines the record types.

When more than one record type is specified for a file, the record identification codes should be coded so that each input record has a unique set of identification codes.

AND Relationship

The AND relationship is used when more than three record identification codes identify a record.

To use the AND relationship, enter at least one record identification code on the first line and enter the remaining record identification codes on the following lines with AND coded in positions 14 through 16 for each additional line used. Positions 7 through 13, 17 through 20, and 42 through 74 of each line with AND in positions 14 through 16 must be blank. Sequence, and record-identifying-indicator entries are made in the first line of the group and cannot be specified in the additional lines.

An unlimited number of AND/OR lines can be used on the input specifications.

OR Relationship

The OR relationship is used when two or more record types have common fields.

To use the OR relationship, enter OR in positions 14 and 15. Positions 7 through 13, 16 through 18, and 42 through 74 must be blank. A record identifying indicator can be entered in positions 19 and 20. If the indicator entry is made and the record identification codes on the OR line are satisfied, the indicator specified in positions 19 and 20 on that line is set on. If no indicator entry is made, the indicator on the preceding line is set on.

An unlimited number of AND/OR lines can be used on the input specifications.

Position 42

Position 42 must be blank.

Field Description Entries

The field description entries (positions 43 through 74) must follow the record identification entries (positions 7 through 42) for each file.

Position 43 (Packed/Binary, Sign Left/Right)

Entry	Explanation
Blank	The input field is in zoned decimal format or is a character field.
P	The input field is in packed decimal format.
B	The input field is in binary format.
L	The numeric input field has a preceding (left) plus or minus sign.
R	The number input field has a following (right) plus or minus sign.

The entry in position 43 specifies the format of the data in the records in the file. This entry has no effect on the format used for internal processing of the input field in the program.

Positions 44-51 (Field Location)

Entry	Explanation
Two 1- to 4-digit numbers	Beginning of a field (from) or end of a field (to).

This entry describes the location and size of each field in the input record. Positions 44 through 47 specify the location of the field's beginning position; positions 48 through 51 specify the location of the field's end position. To define a single-position field, enter the same number in positions 44 through 47 and in positions 48 through 51. Numeric entries must be right-adjusted; leading zeros can be omitted.

The maximum number of positions in the input record for each type of field is as follows:

Number of Positions	Type of Field
30	Zoned decimal numeric (30 digits)
16	Packed numeric (30 digits)
4	Binary (4 or 9 digits)
256	Character (256 characters)
31	Numeric with leading or trailing sign (30 digits)
9999	Data structure.

For arrays, enter the beginning position of the array in positions 44 through 47 and the ending position in positions 48 through 51. The array length must be an integral multiple of the length of an element. The From-To position does not have to account for all the elements in the array. The placement of data into the array starts with the first element.

Position 52 (Decimal Positions)

Entry	Explanation
Blank	Character field
0-9	Number of decimal positions in numeric field.

This entry, used with the packed/binary or left/right sign entry in position 43, describes the format of the field. This entry indicates whether the field described on this line is a character field or a numeric field. If the field is numeric, an entry must be made. The number of decimal positions specified for a numeric field cannot exceed the length of the field. For an array or an array element, this entry must be the same as the entry made on the extension specifications (position 44 or 56) for the array. (This entry can be blank for a numeric array or array element.)

Positions 53-58 (RPG/400 Field Name)

Entry	Explanation
Symbolic name	Field name, data structure name, data structure subfield name, array name, array element, PAGE, PAGE1-PAGE7, *IN, *INxx, or *IN,xx.

These positions name the fields of an input record that are used in an RPG/400 program. This name must follow the rules for symbolic names.

To refer to an entire array on the input specifications, enter the array name in positions 53 through 58. If an array name is entered in positions 53 through 58, control level (positions 59 and 60), matching fields (positions 61 and 62), and field indicators (positions 65 through 70) must be blank.

To refer to an element of an array, specify the array name, followed by a comma, followed by an index. The index is either a numeric field with zero decimal positions or the actual number of the array element to be used. The value of the index can vary from 1 to n, where n is the number of elements within the array.

Positions 59-60 (Control Level)

Entry	Explanation
Blank	This field is not a control field. Control level indicators cannot be used with full procedural files.
L1-L9	This field is a control field.

Positions 59 and 60 indicate the fields that are used as control fields. A change in the contents of a control field causes all operations conditioned by that control level indicator and by all lower level indicators to be processed.

A split control field is a control field that is made up of more than one field, each having the same control level indicator. The first field specified with that control level indicator is placed in the high-order position of the split control field, and the

last field specified with the same control level indicator is placed in the low-order position of the split control field.

Positions 61-62 (Matching Fields)

Entry	Explanation
Blank	This field is not a match field.
M1-M9	This field is a match field.

This entry is used to match the records of one file with those of another or to sequence check match fields within one file. Match fields can be specified only for fields in primary and secondary files.

Match fields within a record are designated by an M1 through M9 code entered in positions 61 and 62 of the appropriate field description specification line. A maximum of nine match fields can be specified.

The match field codes M1 through M9 can be assigned in any sequence. For example, M3 can be defined on the line before M1, or M1 need not be defined at all.

When more than one match field code is used for a record, all fields can be considered as one large field. M1 or the lowest code used is the rightmost or low-order position of the field. M9 or the highest code used is the leftmost or high-order position of the field.

Entries in position 26 (alternate collating sequence) and position 43 (file translation) of the control specification can be used to alter the collating sequence for match fields.

If match fields are specified for only a single sequential file (input, update, or combined), match fields within the file are sequence checked. The MR indicator is not set on and cannot be used in the program. An out-of-sequence record causes the RPG/400 exception/error handling routine to be given control.

In addition to sequence checking, match fields are used to match records from the primary file with those from secondary files.

Positions 63-64 (Field Record Relation)

Entry	Explanation
Blank	The field is common to all record types.
01-99	General indicators.
L1-L9	Control level indicators.
MR	Matching record indicator.
U1-U8	External indicators.
H1-H9	Halt indicators.

Entry	Explanation
RT	Return indicator.

Field record relation indicators are used to associate fields within a particular record type when that record type is one of several in an OR relationship. This entry reduces the number of lines that must be written.

The field described on a line is extracted from the record by the RPG/400 program only when the indicator coded in positions 63 and 64 is on or when positions 63 and 64 are blank. When positions 63 and 64 are blank, the field is common to all record types defined by the OR relationship.

Field record relation indicators can be used with control level fields (positions 59 and 60) and matching fields (positions 61 and 62).

Positions 65-70 (Field Indicators)

Entry	Explanation
Blank	No indicator specified
01-99	General indicators
H1-H9	Halt indicator
U1-U8	External indicators
RT	Return indicator.

Entries in positions 65 through 70 test the status of a field or of an array element as it is read into the program. Field indicators are specified on the same line as the field to be tested. Depending on the status of the field (plus, minus, zero, or blank), the appropriate indicator is set on and can be used to condition later specifications. The same indicator can be specified in two positions, but it should not be used for all three positions. Field indicators cannot be used with non-indexed arrays or look-ahead fields.

Positions 65 and 66 (plus) and positions 67 and 68 (minus) are valid for numeric fields only. Positions 69 and 70 can be used to test a numeric field for zeros or a character field for blanks.

The field indicators are set on if the field or array element meets the condition specified when the record is read. Each field indicator is related to only one record type; therefore, the indicators are not reset (on or off) until the related record is read again or until the indicator is defined in some other specification.

Positions 71-74

Positions 71 through 74 must be blank.

Positions 75-80

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Externally Described Files

Position 6 (Form Type)

An I must appear in position 6 to identify this line as an input specifications statement.

Record Identification Entries

When the description of an externally described file is retrieved by the compiler, the record definitions are also retrieved. To refer to the record definitions, specify the record format name in the input, calculation, and output specifications of the program. Input specifications for an externally described file are required if:

- Record identifying indicators are to be specified.
- A field within a record is to be renamed for the program.
- Control level or matching field indicators are to be used.
- Field indicators are to be used.

The field description specifications must immediately follow the record identification specification for an externally described file.

A record line for an externally described file defines the beginning of the override specifications for the record. All specifications following the record line are part of the record override until another record format name or file name is found in positions 7 through 14 of the input specifications. All record lines that pertain to an externally described file must appear together; they cannot be mixed with entries for other files.

Positions 7-14 (Record Name)

Enter one of the following:

- The external name of the record format. (The file name cannot be used for an externally described file.)
- The RPG/400 name specified by the RENAME option on the file description specifications continuation line if the external record format was renamed. A record format name can appear only once in positions 7 through 14 of the input specifications for a program.

Positions 15-18

Positions 15 through 18 must be blank.

Positions 19-20 (Record Identifying Indicator)

The specification of record identifying indicators in these positions is optional but, if present, follows the rules as described under "Program Described Files" on page 5-10 earlier in this chapter, except for look-ahead specifications, which are not allowed for an externally described file.

Positions 21-41 (Record Identification Code)

Positions 21 through 41 must be blank. Record format names are used to determine the record types used in an externally described file.

Positions 42-74

Positions 42-74 must be blank.

Positions 75-80

Positions 75-80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Field Description Entries

The field description specifications for an externally described file can be used to rename a field within a record for a program or to specify control level, field indicator, and match field functions. The field definitions (attributes) are retrieved from the externally described file and cannot be changed by the program. If the attributes of a field are invalid to an RPG/400 program (such as numeric length greater than 30 digits), the field cannot be used in RPG/400. RPG/400 does diagnostic checking on fields contained in an external record format just as it does for an RPG/400 source statement.

Positions 7-20

Positions 7 through 20 must be blank.

Positions 21-30 (External Field Name)

If a field within a record in an externally described file is to be renamed, enter the external name of the field in these positions. A field may have to be renamed because the external name is longer than 6 characters or because the name is the same as a field name specified in the program and two different names are required.

Positions 31-52

Positions 31 through 52 must be blank.

Positions 53-58 (RPG/400 Field Name)

The field name entry is made only when it is required for the RPG/400 function (such as control levels) added to the external description. The field name entry contains one of the following:

- The name of the field as defined in the external record description (if 6 characters or less).
- The name specified to be used in the program that replaced the external name specified in positions 21 through 30.

The field name must follow the rules for using symbolic names.

Positions 59-60 (Control Level)

This entry indicates whether the field is to be used as a control field in the program.

Entry	Explanation
Blank	This field is not a control field.
L1-L9	This field is a control field.

Note: For externally described files, split control fields are combined in the order in which the fields are specified on the data description specifications (DDS), not in the order in which the fields are specified on the input specifications.

Positions 61-62 (Matching Fields)

This entry indicates whether the field is to be used as a match field.

Entry	Explanation
Blank	This field is not a match field.
M1-M9	This field is a match field.

See "Positions 61-62 (Matching Fields)" on page 5-19 for more information on match fields.

Positions 63-64

Positions 63 and 64 must be blank.

Positions 65-70 (Field Indicators)

Entry	Explanation
Blank	No indicator specified
01-99	General indicators
H1-H9	Halt indicators
U1-U8	External indicators
RT	Return indicator.

Positions 71-74

Positions 71 through 74 must be blank.

Positions 75-80

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Data Structure Specifications

A data structure can be used to:

- Divide a field into subfields without using the MOVE or MOVEL operations.
- Operate on a subfield and change the contents of a subfield.
- Redefine the same internal area more than once using different data formats.

Data structures are defined on the input specifications the same way records are defined. The record specification line contains the data structure statement (DS in positions 19 and 20) and the data structure name (optional). The field specification lines contain the subfield specifications for the data structure.

Data structure specifications must follow the input specifications for records. All entries describing a data structure and its subfields must appear together.

Data Structure Statement Specifications

Position 6 (Form Type)

Position 6 must contain an I for input specifications.

Positions 7-12 (Data Structure Name)

Positions 7 through 12 can contain the name of the data structure being defined. The data structure name is optional, and is limited to 6 characters. A data structure name must follow the rules for using symbolic names. A data structure name can be specified anywhere a character field can be specified.

Positions 13-16

Positions 13 through 16 must be blank.

Position 17 (Number)

Entry	Explanation
Blank	Subfield definitions for this data structure follow this specification.

Entry	Explanation
E	Subfield definitions are described externally. Positions 7 through 12 must contain the name of an externally described file if positions 21 through 30 are blank. The file name must be limited to 6 characters.

Position 18 (Option)

Entry	Explanation
Blank	This data structure is not a program status or data area data structure.
S	This data structure is the program status data structure. Only one data structure can be specified as the program status data structure.
U	This is a data area data structure. RPG/400 retrieves the external data area (named in positions 7 through 12) when the program starts and rewrites it when the program ends. If you put blanks in positions 7 through 12, RPG/400 uses the local data area.

Positions 19-20 (Record Identifying Indicator)

Positions 19 and 20 must contain DS to indicate a data structure.

Positions 21-30 (External Name)

Entry	Explanation
Blank	The data structure subfields are defined in the program.
File name	This is the name of the file whose first record format contains the field descriptions used as the subfield descriptions for this data structure.

Positions 31-43

Positions 31 through 43 must be blank.

Positions 44-47 (Occurs N Times)

Entry	Explanation
Blank	This is not a multiple-occurrence data structure.
1-9999	The number (right-adjusted) indicating the number of occurrences of a multiple-occurrence data structure.

These positions must be blank if the data structure is the program status data structure (indicated by an S in position 18), a file information data structure (INFDS), or a data area data structure.

Positions 48-51 (Length)

The length of the data structure can be specified in positions 48 through 51. This entry is optional but, if used, must be right-adjusted. If this entry is not made, the length of the data structure is one of the following:

- The length specified on the input field specifications if the data structure name is an input field.
- The highest To position specified for a subfield within the data structure if the data structure name is not an input field.

Positions 52-74

Positions 52 through 74 must be blank.

Positions 75-80

Positions 75 through 80 may be used for comments or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Data Structure Subfield Specifications

Specifications for subfields, if used, must follow the data structure specification statement to which they apply.

Positions 7-20

Positions 7 through 20 must be blank.

Positions 21-30 (External Field Name)

To rename a subfield in an externally described data structure, specify the external name in positions 21 through 30, and specify the name to be used in the program in positions 53 through 58. The remaining positions must be blank.

Positions 31-42

Positions 31 through 42 must be blank.

Position 43 (Packed/Binary, Sign Left/Right)

Entry	Explanation
Blank	Subfield is in zoned decimal format or is character data.
P	Subfield is in packed decimal format.
B	Subfield is in binary format.

Positions 44-51 (Field Location)

Entry	Explanation
Two 1- to 4-digit numbers	Beginning of a subfield (from) and end of a subfield (to).

Entry	Explanation
Keywords	For the program status data structure or for a file information data structure, special keywords define the location of the subfields in the data structures.

Position 52 (Decimal Positions)

Entry	Explanation
Blank	Character subfield
0-9	Number of decimal positions in a numeric subfield.

Position 52, along with position 43, describes the format of the subfield. An entry must be made in position 52 for a numeric subfield.

Positions 53-58 (RPG/400 Field Name)

In positions 53 through 58, enter the name of the subfield that is being defined.

Positions 59-74

Positions 59 through 74 must be blank.

Positions 75-80

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiler listing.

Chapter 6. Calculation Specifications

Calculation specifications indicate the operations to be done on the data in a program. Two general rules govern the writing of calculation entries:

- Each operation is specified on one line, except when there are AND/OR lines in the calculation.
- Calculation entries must be grouped in the following order:
  - Detail calculations
  - Total calculations
  - Subroutines.

Calculations within the groups must be specified in the order in which they are to be done.

Each calculation specifications statement is divided into three parts that specify the following:

- When calculations are to be done. The conditioning indicators specified in positions 7 through 17 determine when the calculations are to be done and under what conditions they are to be done.
- What kind of calculations are to be done. The entries specified in positions 18 through 53 determine the kind of calculations to be done, specify the data (such as fields or files) upon which the operation is to be done, and specify the field that is to contain the results of the calculation.
- What tests are to be made on the results of the operation. Indicators specified in positions 54 through 59 are used to test the results of the calculations and can condition subsequent calculations or output operations. The resulting indicator positions have various uses, depending on the operation code. For the uses of these positions, see the individual operation codes in Chapter 9, "Operation Codes."

The calculation specifications are entered on the RPG/400 Calculation Specifications (see Figure 6-1 on page 6-2). See Chapter 9, "Operation Codes" for details on how these positions must be specified for individual calculation operations.

The calculation specification can also be used to enter SQL statements into an RPG/400 program. See *RPG/400 User's Guide* and *SQL/400 Reference* for more information.

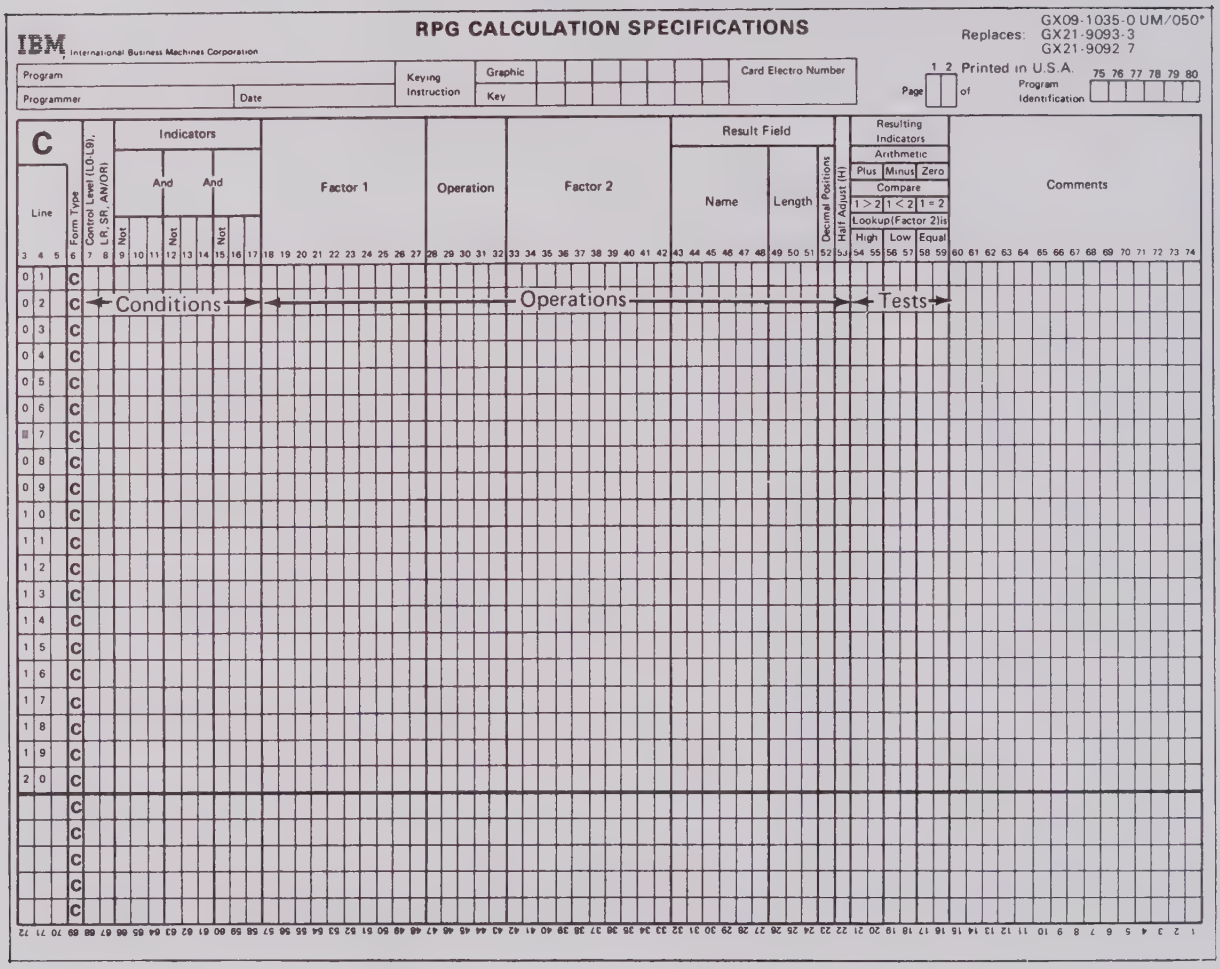


Figure 6-1. RPG/400 Calculation Specifications Form

Calculation Specification Summary Chart

Table 6-1 (Page 1 of 3). Calculation Specification Summary Chart

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specifications form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	C	Identification for a calculation specification.
7-8	Control level	Blank	The calculation operation is done at detail calculation time of each program cycle if the indicators in positions 9 through 17 allow it, if the calculation is part of a subroutine, or if the calculation is a declarative statement.
		L0	The calculation operation is done at total calculation time of each program cycle if the indicators in positions 9 through 17 allow it.
		L1-L9	The calculation operation is done at total calculation time if the indicator is on (because a control break occurs or because the indicator is set on) and if the indicators in positions 9 through 17 allow it.
		LR	The calculation operation is done after the last record has been processed or after the LR indicator has been set on if the indicators in positions 9 through 17 allow it.
		SR	The calculation operation is part of an RPG/400 subroutine.
		AN, OR	Indicators on more than one line condition the calculation.

Table 6-1 (Page 2 of 3). Calculation Specification Summary Chart

Positions	Name	Entry	Explanation
9-17	Indicators	Blank	The operation is done if the condition specified in positions 7 and 8 is satisfied.
			The operation is done. if the condition specified in positions 7 and 8 is satisfied and if the conditioning indicators in positions 9 through 17 are satisfied. The following indicators are valid in positions 9 through 17:
		01-99	• General indicator
		KA-KN, KP-KY	• Function key indicator
		L1-L9	• Control level indicator
		LR	• Last record indicator
		MR	• Matching record indicator
		H1-H9	• Halt indicator
		RT	• Return indicator
		U1-U8	• External indicator
		OA-OG, OV	• Overflow indicator.
18-27	Factor 1	Symbolic name or literal	Entry specifies a symbolic name or actual data on which an operation is to be done. Valid entries depend on the operation code.
28-32	Operation	Operation code	Entry specifies the operation to be done.
33-42	Factor 2	Symbolic name or literal	Entry specifies a symbolic name or actual data on which an operation is to be done. Valid entries depend on the operation code.
43-48	Result field	Field name	The result field names the field that contains the result of the calculation operation specified in positions 28 through 32.
49-51	Field length	Blank	The result field is defined elsewhere.
		1-30	Numeric field length.
		1-256	Character field length.

Table 6-1 (Page 3 of 3). Calculation Specification Summary Chart			
Positions	**Name**	**Entry**	**Explanation**
52	Decimal positions	Blank	The result field is character data or has been defined elsewhere in the program.
		0-9	Number of decimal positions in a numeric result field.
53	Half-adjust	Blank	Half-adjust is not done.
		H	Half-adjust is done.
54-59	Resulting indicators	Blank	No resulting indicator.
		01-99	General indicators.
		KA-KN, KP-KY	Function key indicators.
		H1-H9	Halt indicators.
		L1-L9	Control level indicators.
		LR	Last record indicator.
		OA-OG, OV	Overflow indicators.
		U1-U8	External indicators.
		RT	Return indicator.
			Note: The resulting indicator positions (54 and 55, 56 and 57, and 58 and 59) have different uses, depending on the operation code specified.
60-74	Comments	Comments	These positions can be used for comments to document the purpose of the calculation.
75-80		Optional	This space is available for comments.

Calculation Specification Statement

Position 6 (Form Type)

A C must appear in position 6 to identify this line as a calculation specifications statement.

Positions 7-8 (Control Level)

Entry	Explanation
Blank	The calculation operation is done at detail calculation time for each program cycle if the indicators in positions 9 through 17 allow it; or the calculation is part of a subroutine.
L0	The calculation operation is done at total calculation time for each program cycle.
L1-L9	The calculation operation is done when the appropriate control break occurs at total calculation time, or when the indicator is set on.
LR	The calculation operation is done after the last record has been processed or after the LR indicator has been set on.
SR	The calculation operation is part of an RPG/400 subroutine. A blank entry is also valid for calculations that are part of a subroutine.
AN, OR	Indicators on more than one line condition the calculation.

Control Level Indicators

The L0 entry is used in positions 7 and 8 to indicate that the calculation is to be done during total time and is not dependent on the occurrence of a control break.

If indicators L1 through L9 are specified in positions 7 and 8, the calculation is processed at total time only when the specified indicator is on. Remember that, if L1 through L9 are set on by a control break, all lower level indicators are also set on. If positions 7 and 8 are blank, the calculation is done at detail time, is a statement within a subroutine, or is a declarative statement.

The following operations can be specified within total calculations with positions 7 and 8 blank: PLIST, PARM, KLIST, KFLD, TAG, DEFN, and ELSE. (Conditioning indicators in positions 9 through 17 are not allowed with these operations.) In addition, all the preceding operations except TAG and ELSE can be specified anywhere within the calculations, even between an ENDSR operation of one subroutine and the BEGSR operation of the next subroutine or after the ENDSR operation for the last subroutine.

Last Record Indicator

The LR Indicator, if specified in positions 7 and 8, causes the calculation to be done during the last total time.

If there is a primary file but no secondary files in the program, RPG/400 sets the LR indicator on after the last input record has been read, the calculations specified for the record have been done, and the detail output for the last record read has been completed.

If there is more than one input file (primary and secondary), the RPG/400 programmer determines which files are to be checked for end-of-file by entering an E in position 17 of the file description specifications. RPG/400 sets LR on when all files with an end-of-file specification (E in position 17 of the file description specifications) have been completely read, when detail output for the last record in these files has been completed, and after all matching secondary records have been processed.

When the LR indicator is set on after the last input record has been read, the control level indicators L1 through L9 are also set on. If the indicators L1 through L9 have not been defined by an entry in positions 59 and 60 of the input specifications or by *INxx when used in a result field, the indicators are set on when LR is on, but they cannot be used in other specifications.

Subroutine Identifier

An SR entry in positions 7 and 8 indicates that the specification is part of a subroutine. The SR entry is not required. Subroutine lines must appear after the total calculation specifications. The operation codes BEGSR and ENDSR serve as delimiters for a subroutine.

AND/OR Lines Identifier

Positions 7 and 8 can contain AN or OR to define additional indicators (positions 9 through 17) for a calculation. Indicators in positions 9 through 17 contained in one line are always in an AND relationship. Indicators contained on more than one line can be a combination of AND and OR relationships. A maximum of seven AN/OR lines can be specified in one group.

The entry in positions 7 and 8 of the line immediately preceding an AN/OR line or a group of AN/OR lines determines when the calculation is to be processed. The entry in positions 7 and 8 on the first line of a group applies to all AN/OR lines in the group. A control level indicator (L1 through L9, L0, or LR) is entered for total calculations, an SR or blanks for subroutines, and a blank for detail calculations.

Positions 9-17 (Indicators)

Entry	Explanation
Blank	The operation is processed on every record if the condition specified in positions 7 and 8 is satisfied.
01-99	General indicators.
KA-KN, KP-KY	Function key indicators.

Entry	Explanation
L1-L9	Control level indicators.
LR	Last record indicator.
MR	Matching record indicator.
H1-H9	Halt indicators.
RT	Return indicator.
U1-U8	External indicators.
OA-OG, OV	Overflow indicator.

Positions 10 and 11, 13 and 14, and 16 and 17 contain indicators that are tested to determine if a particular calculation is to be processed. A blank in positions 9, 12, and 15 designates that the indicator must be on for a calculation to be done. An N in positions 9, 12, and 15 designates that the associated indicator must be off for a calculation to be done.

One to three indicators can be entered in positions 9 through 17 on one line. Indicators on the same line are in an AND relationship. The AND relationship means that all three indicator conditions must be satisfied before the calculation can take place. If fewer than three indicators are specified, entries need not be made in sequence; that is, an indicator can be specified in positions 16 and 17, and positions 10 through 15 can be blank. When more than three indicators are needed to condition a calculation, AN/OR must be used. Positions 9 through 17 can contain a combination of the type of indicators discussed in the preceding text.

Positions 18-27 (Factor 1)

Factor 1 names a field or gives actual data (literals) on which an operation is to be done. The entry must begin in position 18. The entries that are valid for factor 1 depend on the operation code specified in positions 28 through 32. For the specific entries for factor 1 for a particular operation code, see Chapter 9, "Operation Codes."

Literals

Literals can be specified in factor 1 or factor 2 of certain operations. A literal is a self-defining constant that can be referred to in a program. A literal can be either character data or numeric data.

Character Literals

Consider the following when using a character literal:

- Any combination of characters can be used in a character literal. Embedded blanks are valid.
- If a character literal is specified in factor 1 or factor 2, the maximum length of the literal is 8 characters.
- Character literals must be enclosed in apostrophes (').

- An apostrophe required as part of a literal is represented by two apostrophes. For example, the literal O'CLOCK is coded as 'O''CLOCK'..
- Character literals cannot be used for arithmetic operations.

Numeric Literals

Consider the following when using a numeric literal:

- A numeric literal consists of any combination of the digits 0 through 9. A decimal point or a sign can be included.
- The sign (+ or -), if present, must be the leftmost character. An unsigned literal is treated as a positive number.
- The maximum length of a numeric literal is 10 characters, including the sign and the decimal point.
- Blanks cannot appear in a numeric literal.
- Numeric literals must not be enclosed in apostrophes (').
- Numeric literals are used in the same way as a numeric field.
- The character (comma or period) used for the decimal notation is determined by the inverted print option specified in position 21 of the control specification.

Positions 28-32 (Operation)

Positions 28 through 32 specify the kind of operation to be done using factor 1, factor 2, and the result field entries. The operation code must begin in position 28. For further information on the operation codes, see Chapter 9, "Operation Codes."

The program processes the operations in the order specified on the calculation specifications form.

Positions 33-42 (Factor 2)

Factor 2 names a field or gives the actual data (literals) on which a calculation is to be done. For the file operation codes, factor 2 names a file or record format to be used. The entry must begin in position 33. The entries that are valid for factor 2 depend on the operation code specified in positions 28 through 32. For the specific entries for factor 2 for a particular operation code, see Chapter 9, "Operation Codes."

Positions 43-48 (Result Field)

The result field names the field that contains the result of the calculation operation specified in positions 28 through 32. Array elements are treated as fields. A look-ahead field, UDATE, UYEAR, UMONTH, or UDAY cannot appear as the result field of a calculation. See Chapter 9, "Operation Codes" for the result field rules for individual operation codes.

Positions 49-51 (Field Length)

Entry	Explanation
1-30	Numeric field length.
1-256	Character field length.
Blank	The result field is defined elsewhere.

Positions 49 through 51 specify the length of the result field. This entry is optional, but can be used to define a field not defined elsewhere in the program. These definitions of the field entries are allowed if the result field contains a field name.

The entry specifies the number of positions to be reserved for the result field. The entry must be right-adjusted. The unpacked length (number of digits) must be specified for numeric fields.

If the length of an arithmetic result to the left of the decimal point exceeds the specified length, the excess leftmost digits and any excess decimal positions are dropped. No diagnostic message occurs if the result is truncated.

If the result field is defined elsewhere in the program, no entry is required for the length. However, if the length is specified, and if the result field is defined elsewhere, the length must be the same as the previously defined length.

If half-adjustment is specified in position 53 of the calculation specifications, the entries for field length (positions 49 through 51) and decimal positions (position 52) refer to the length of the result field after half-adjustment.

Position 52 (Decimal Positions)

Entry	Explanation
Blank	The result field is character data or has been defined elsewhere in the program.
0-9	Number of decimal positions in a numeric result field.

Position 52 indicates the number of positions to the right of the decimal in a numeric result field. If the numeric result field contains no decimal positions, enter a 0 (zero). This position must be blank if the result field is character data. This position can be left blank if the result field is numeric but was described by input or calculation specifications or in an external description. In this case, field length (positions 49 through 51) must also be left blank. The number of decimal positions specified cannot exceed the length of the field.

Position 53 (Half Adjust)

Entry	Explanation
Blank	Half adjust is not done.
H	Half adjust.

Position 53 indicates whether the contents of the result field are to be half adjusted (rounded). Half-adjusting is done by algebraically adding 5 (-5 if the field is negative) one position to the right of the last specified decimal position in the result field. The half-adjust entry is allowed only with arithmetic operations, but not with an MVR operation or with a DIV operation followed by the MVR operation. Half adjust can be specified only if the number of decimal positions in the generated result is greater than the number of decimal positions in the result field. Resulting indicators are set according to the value of the result field after half-adjusting has been done.

Positions 54-59 (Resulting Indicators)

These positions can be used, for example, to test the value of a result field after the completion of an operation, or to indicate an end-of-file, error, or record-not-found condition. The resulting indicator positions have different uses, depending on the operation code specified. See the individual operation codes in Chapter 9, "Operation Codes" for a description of the associated resulting indicators. For arithmetic operations, the result field is tested only after the field is truncated and half-adjustment is done (if specified). The setting of indicators depends on the results of the tests specified.

Entry	Explanation
Blank	No resulting indicator specified
01-99	General indicators
KA-KN, KP-KY	Function key indicators
H1-H9	Halt indicators
L1-L9	Control level indicators
LR	Last record indicator
OA-OG, OV	Overflow indicators
U1-U8	External indicators
RT	Return indicator.

Resulting indicators cannot be used when the result field uses a nonindexed array.

If the same indicator is used as a resulting indicator on more than one calculation specification, the last calculation specification processed determines the status of that indicator.

Remember the following points when specifying resulting indicators:

- When the calculation operation is done, the specified resulting indicators are set to off, and, if a condition specified by a resulting indicator is satisfied, that indicator is set to on.
- When a control level indicator (L1 through L9) is set on, the lower level indicators are not set on.
- When a halt indicator (H1 through H9) is set on, the program ends unless the halt indicator is set to off.

Positions 60-74 (Comments)

Positions 60 through 74 of each calculation specification line can be used for comments to document the purpose of that calculation.

Positions 75-80

Positions 75 through 80 can be used for comments, or left blank. These positions are not printed contiguously with positions 6-74 on the compiled listing.

Chapter 7. Output Specifications

Output specifications describe the record and the format of fields in a program described output file and when the record is to be written. Output specifications are optional for an externally described file. Output specifications can be divided into two categories: record identification and control (positions 7 through 37), and field description and control (positions 23 through 70). These specifications are entered on the RPG/400 Output Specifications (see Figure 7-1 on page 7-2).

This chapter is organized in the following sequence:

- Output specifications summary charts
- Entries for program described files
- Entries for externally described files.

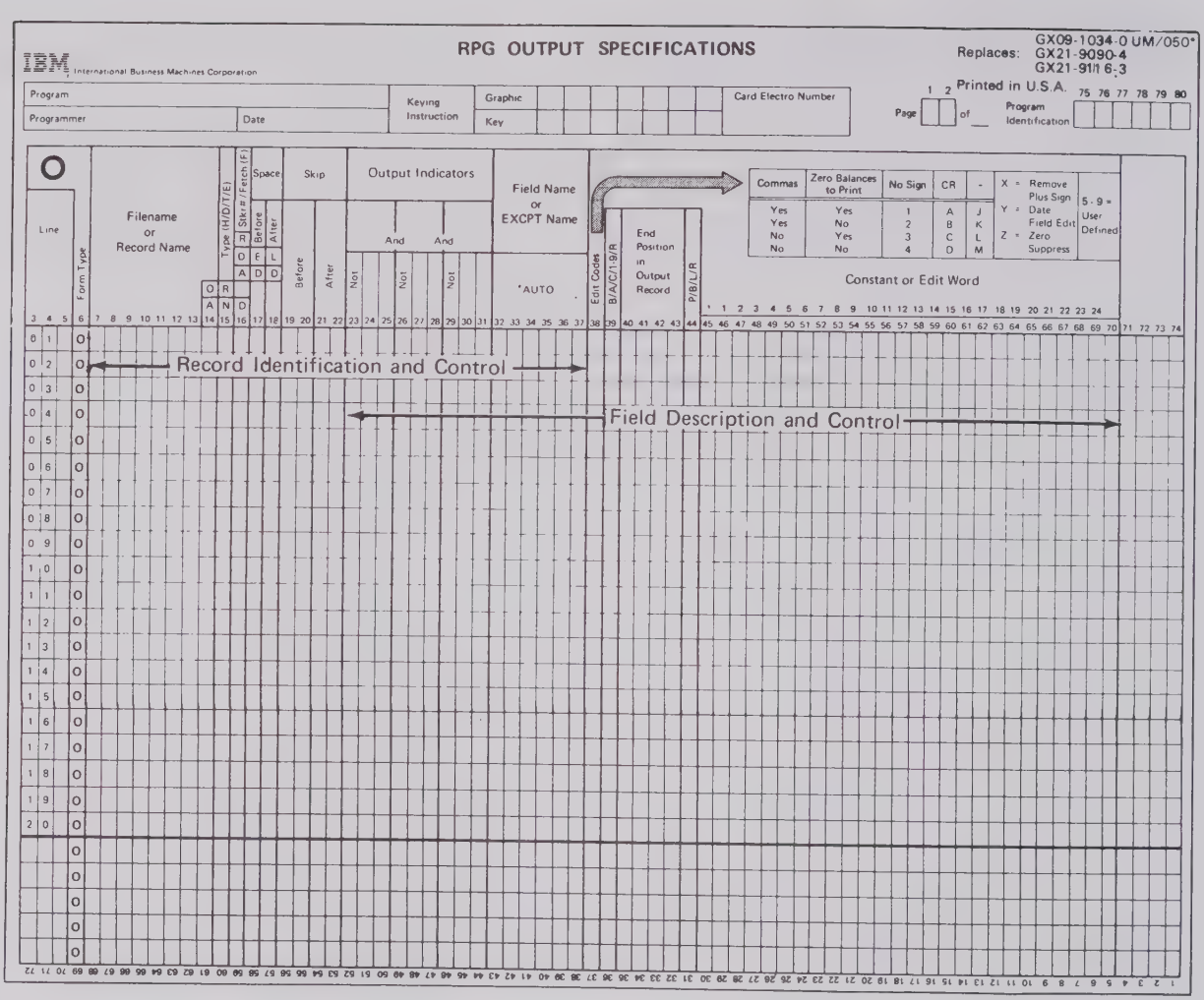


Figure 7-1. RPG/400 Output Specifications Form

Output Specifications Summary Charts

Program Described Files, Record Identification and Control Entries (Record Line)

Table 7-1 (Page 1 of 3). Program Described Files, Record Identification and Control Entries (Record Line)

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers the specification line.
6	Form type	O	Identification for an output specification.
7-14	File name	Valid file name	Same file name that appears on the file description specifications for the output file.
14-16	AND/OR	AND or OR	AND/OR indicates a relationship between lines of output indicators. Not valid for output fields.
15	Type	H or D	Detail records.
		T	Total records.
		E	Exception records.
16-18	ADD/DEL	ADD	Add a record to the file.
		DEL	Delete a record from a file.
1.16	Fetch over-flow	Blank	If this position is blank for a printer file, overflow is not fetched.
		F	Fetch overflow for printer files.
	Release	R	Release a device (work station) after output.
17	Space before	0 or blank	Spaces zero before the line is printed.
		1	Spaces one before the line is printed.
		2	Spaces two before the line is printed.
		3	Spaces three before the line is printed.
18	Space after	0 or blank	Spaces zero after the line is printed.
		1	Spaces one after the line is printed.
		2	Spaces two after the line is printed.
		3	Spaces three after the line is printed.

Table 7-1 (Page 2 of 3). Program Described Files, Record Identification and Control Entries (Record Line)

Positions	Name	Entry	Explanation
19-20	Skip before	01-99	Skip to specified line number before printing line.
		A0-A9	Specifies numbers between 100 and 109. Skip to specified line number before printing line.
		B0-B2	Specifies numbers between 110 and 112. Skip to specified line number before printing line.
21-22	Skip after	01-99	Skip to specified line number after printing the line.
		A0-A9	Specifies numbers between 100 and 109. Skip to specified line number after printing line.
		B0-B2	Specifies numbers between 110 and 112. Skip to specified line number after printing line.
23-31	Output indicators	Blank	The line or field is written every time the type of record is checked for output.
			The line or field is written if the indicators in these positions are satisfied. The following indicators are valid in these positions:
		01-99	A general indicator used as a resulting indicator, field indicator, or record identifying indicator
		KA-KN, KP-KY	Function key indicator
		L1-L9	Control level indicator
		H1-H9	Halt indicator
		U1-U8	External indicator set before running the program or set as a result of a calculation operation
		OA-OG, OV	Overflow indicator previously assigned to this file
		MR	Matching record indicator
		LR	Last record indicator
		RT	Return indicator
		1P	First-page indicator (valid only on heading or detail lines).

Table 7-1 (Page 3 of 3). Program Described Files, Record Identification and Control Entries (Record Line)			
Positions	**Name**	**Entry**	**Explanation**
32-37	EXCPT name	Record group name	A name placed in these positions specifies exception records to be written. Any number of exception output records can use the same EXCPT name, and they do not have to be consecutive.
38-74		Blank	
75-80			This space is available for comments.

Program Described Files, Field Description and Control Entries (Field Line)

Table 7-2 (Page 1 of 2). Program Described Files, Field Description and Control Entries (Field Line)			
Positions	**Name**	**Entry**	**Explanation**
7-22		Blank	
23-31	Field output indicators	See output indicators	Used to indicate whether fields within records are printed.
9.32-37	Field name	Valid field name	Any field name previously defined in the program.
		PAGE1, PAGE1-PAGE7	Automatic page numbering. Field output indicators control the resetting of the PAGE field to zero.
		UDATE, UDAY, UMONTH, UYEAR	Allows the programmer to supply a date for the program at run time. UDATE places the full date in the output record; UDAY, only the day; UMONTH, only the month; UYEAR, only the year.
		*PLACE	Allows programmer to repeat data in an output record.
38	Edit code	Blank	No edit code is used.
		1-9, A-D, J-M, X, Y, Z	Numeric fields are zero-suppressed and punctuated according to a predefined pattern without the use of edit words. Edit codes 5 through 9 are defined externally to RPG/400 by an OS/400 function.
39	Blank after	Blank	The field is not reset.
		B	The field specified in positions 32 through 37 is reset to blank or zero after the output operation is complete.

Table 7-2 (Page 2 of 2). Program Described Files, Field Description and Control Entries (Field Line)

Positions	Name	Entry	Explanation
40-43	End position in output record	Blanks, ±nnn, nnnn	Defines the end position of a field or constant in the output record. The end position must not exceed the record length for the file.
		K1-K8	Length of the format name for a WORKSTN file.
44	Packed/binary,	Blank	The field is to be written in zoned decimal or character format; numeric data is edited or a constant is specified on this line.
		P	The field is to be written in packed decimal format.
		B	The field is to be written in binary format.
	sign left/right	L	The numeric output field is to have a preceding (left) plus or minus sign.
		R	The numeric output field is to have a following (right) plus or minus sign.
45-70	Constant or edit word	Constant or edit word	Enter a constant or an edit word.
		Format name	Name of the external format to be used by the WORKSTN file.
70-74		Blank	
75-80		Optional	This space is available for comments.

Externally Described Files, Record Identification and Control Entries

Table 7-3 (Page 1 of 3). Externally Described Files, Record Identification and Control Entries

Positions	Name	Entry	Explanation
1-2	Page	Page number	Entry assigns a page number to each specification form.
3-5	Line	Line number	Entry numbers each specification line.
6	Form type	O	Identification for an output specification.
7-14	Record name	Valid record format name	Enter the record format name for externally described files.
14-16	AND/OR	AND or OR	AND/OR specifies a relationship between two lines of output indicators.

Table 7-3 (Page 2 of 3). Externally Described Files, Record Identification and Control Entries

Positions	Name	Entry	Explanation
15	Type	H or D	Detail records.
		T	Total records.
		E	Exception records.
16	Release	R	Release a device after output.
16-18	ADD/DEL	ADD	Add a record to the file.
		DEL	Delete an existing record from the file.
16-22	Space/Skip, Fetch overflow	Blank	Not used for externally-described files.
23-31	Output indicators	Blank	The line or field is written every time the type of record is checked for output.
			The line or field is written if the indicators in these positions are satisfied. The following indicators are valid in these positions:
		01-99	A general indicator used as a resulting indicator, field indicator, or record identifying indicator
		KA-KN, KP-KY	Function key indicator
		L1-L9	Control level indicator
		H1-H9	Halt indicator
		U1-U8	External indicator set before running the program or set as a result of a calculation operation
		OA-OG, OV	Overflow indicator previously assigned to this file
		MR	Matching record indicator
		LR	Last record indicator
		RT	Return indicator
		1P	First-page indicator (valid only on heading or detail lines).

Table 7-3 (Page 3 of 3). Externally Described Files, Record Identification and Control Entries			
Positions	**Name**	**Entry**	**Explanation**
32-37	EXCPT name	Record group name	A name placed in these positions specifies exception records to be written. Any number of exception output records can use the same EXCPT name, and they do not have to be consecutive.
38-74		Blank	
75-80		Optional	This space is available for comments.

Externally Described Files, Field Description and Control Entries

Table 7-4. Externally Described Files, Field Description and Control Entries			
Positions	**Name**	**Entry**	**Explanation**
7-22		Blank	
23-31	Field output indicators	See output indicators	Used to indicate whether fields within records are written.
32-37	Field name	Valid field name	Either the externally described field name or the new name as renamed using positions 53-58 of the input specifications.
		*ALL	All fields are written.
38		Blank	
39	Blank after	Blank	The field is not reset.
		B	The field specified in positions 32 through 37 is reset to blank or zero after the output operation is complete.
40-74		Blank	
75-80		Optional	This space is available for comments.

Program Described Files

Position 6 (Form Type)

An O must appear in position 6 to identify this line as an output specifications statement.

Record Identification and Control Entries

Entries in positions 7 through 37 identify the output records that make up the files, provide the correct spacing on printed reports, and determine under what conditions the records are to be written.

Positions 7-14 (File Name)

Entry	Explanation
A valid file name	Same file name that appears on the file description specifications for the output file.

Specify the file name on the first line that defines an output record for the file. The file name specified must be the same file name assigned to the output, update, or combined file on the file description specifications. If records from files are interspersed on the output specifications, the file name must be specified each time the file changes.

For files specified as output, update, combined, or input with ADD, at least one output specification is required unless an explicit file operation code with a data structure name specified in the result field is used in the calculations. For example, a WRITE operation does not require output specifications.

Positions 14-16 (AND/OR)

Entry	Explanation
AND or OR	AND/OR indicates a relationship between lines of output indicators. AND/OR lines are valid for output records, but not for fields.

Positions 14 through 16 specify AND/OR lines for output operations. To specify this relationship, enter AND/OR in positions 14 through 16 on each additional line following the line containing the file name. At least one indicator must be specified on each AND line. For an AND relationships and fetch overflow (position 16) must be specified on the first line only (file name line). A fetch overflow entry is required on OR lines for record types requiring the fetch overflow routine.

Positions 17 through 22 (spacing and skipping) must be blank on an AND line. In an OR relationship, positions 17 through 22 can be used; if they are blank, the definitions from the preceding line are used. Positions 7 through 13 must be blank when AND/OR is specified.

An unlimited number of AND/OR lines can be specified on the output specifications.

Position 15 (Type)

Entry	Explanation
H or D	Detail records usually contain data that comes directly from the input record or that is the result of calculations processed at detail time. Heading records usually contain constant identifying information such as titles, column headings, page number, and date. RPG/400 makes no distinction between heading and detail records. The H/D specifications are available to help the programmer document the program.
T	Total records usually contain data that is the end result of specific calculations on several detail records.
E	Exception records are written during calculation time. Exception records can be specified only when the operation code EXCPT is used. See Chapter 9, "Operation Codes" for further information on the EXCPT operation code.

Position 15 indicates the type of record to be written. Position 15 must have an entry for every output record. Heading (H) and detail (D) lines are both processed as detail records. No special sequence is required for coding the output records; however, all lines of each record type (H, D, T, or E) are handled at separate times within the program cycle.

Positions 16-18 (ADD/DEL)

Entry	Explanation
ADD	Add a record to the file.
DEL	Delete an existing record from the file. The deleted record cannot be retrieved; the record is deleted from the system.

An entry of ADD is valid for input, output, or update files. DEL is valid for update files only.

The device specified in positions 40 through 46 of the file description specifications for the file must be DISK.

Note: The file description specifications for a file using the ADD function must have an A in position 66.

This entry must appear on the same line that contains the record type (H, D, T, E) specification (position 15). If an OR line is used following an ADD or DEL entry, this entry applies to the OR line also.

Position 16 (Fetch Overflow/Release)

Entry	Explanation
Blank	Must be blank for all files except printer files (PRINTER specified in positions 40 through 46 of the file description specifications). If position 16 is blank for printer files, overflow is not fetched.
F	Fetch overflow.
R	Release.

Fetch Overflow

An F in position 16 specifies fetch overflow for the printer file defined on this line. This file must be a printer file that has overflow lines. Fetch overflow is processed only when an overflow occurs and when all conditions specified by the indicators in positions 23 through 31 are satisfied. An overflow indicator cannot be specified on the same line as fetch overflow.

If an overflow indicator has not been specified in positions 33 and 34 of the file description specifications for a printer file, the compiler assigns one to the file. An overflow line is generated by the compiler for the file, except when no other output records exist for the file or when the printer uses externally described data. This compiler-generated overflow can be fetched.

Overflow lines can be written during detail, total, or exception output time. When the fetch overflow is specified, only overflow output associated with the file containing the processed fetch is output. The fetch overflow entry (F) is required on each OR line for record types that require the overflow routine. The fetch overflow routine does not automatically advance forms.

The form length and overflow line can be specified on the line counter specifications, through an OS/400 override command, or in the printer device file.

Release

After an output operation is complete, the device used in the operation is released if you have specified an R in position 16 of the corresponding output specifications. See the REL (Release) operation in Chapter 9, "Operation Codes" for further information on releasing devices.

Positions 17-22 (Space and Skip)

Use positions 17 through 22 to specify line spacing and skipping for a printer file. Spacing refers to advancing one line at a time, and skipping refers to jumping from one print line to another.

If spacing and skipping are specified for the same line, the spacing and skipping operations are processed in the following sequence:

Skip before
Space before
Print a line
Skip after
Space after.

If PRTCTL (printer control option) is not specified on the file description specifications, an entry must be made in one of the following positions when the device is PRINTER: 17 (space before), 18 (space after), 19 and 20 (skip before), or 21 and 22 (skip after). If a space/skip entry is left blank, the particular function with the blank entry (such as space before or space after) does not occur. If entries are made in position 17 (space before) or in positions 19 through 22 (skip before and skip after) and no entry is made in position 18 (space after), no space occurs after printing. When PRTCTL is specified, it is used only on records with blanks specified in positions 17 through 22.

If a skip before or a skip after a line on a new page is specified, but the printer is on that line, the skip does not occur.

Position 17 (Space Before)

Entry	Explanation
0 or blank	No spacing
1	Single spacing
2	Double spacing
3	Triple spacing.

Position 18 (Space After)

Entry	Explanation
0 or blank	No spacing
1	Single spacing
2	Double spacing
3	Triple spacing.

Positions 19-20 (Skip Before)

Entry	Explanation
Blank	No skipping occurs.
01-99	Skip to lines 01 to 99 before printing for printer files. The leading zero is optional.
A0-A9	Skip to lines 100 to 109 before printing for printer files.
B0-B2	Skip to lines 110 to 112 before printing for printer files.

Positions 21-22 (Skip After)

Entry	Explanation
Blank	No skipping occurs.
01-99	Skip to lines 01 to 99 after printing for printer files. The leading zero is optional.

Entry	Explanation
A0-A9	Skip to lines 100 to 109 after printing for printer files.
B0-B2	Skip to lines 110 to 112 after printing for printer files.

Positions 23-31 (Output Indicators)

Entry	Explanation
Blank	The line or field is output every time the record (heading, detail, total, or exception) is checked for output.
01-99	A general indicator that is used as a resulting indicator, field indicator, or record identifying indicator.
KA-KN, KP-KY	Function key indicators.
L1-L9	Control level indicators.
H1-H9	Halt indicators.
U1-U8	External indicator set before running the program or set as a result of a calculation operation.
OA-OG, OV	Overflow indicator previously assigned to this file.
MR	Matching record indicator.
LR	Last record indicator.
RT	Return indicator.
1P	First-page indicator. Valid only on heading or detail lines.

Conditioning indicators are not required on output lines. If conditioning indicators are not specified, the line is output every time that record is checked for output. Up to three indicators can be entered on one specification line to control when a record or a particular field within a record is written. The indicators that condition the output are coded in positions 24 and 25, 27 and 28, and 30 and 31. When an N is entered in positions 23, 26, or 29, the indicator in the associated position must be off for the line or field to be written. Otherwise, the indicator must be on for the line or field to be written.

If more than one indicator is specified on one line, all indicators are considered to be in an AND relationship.

If the output record must be conditioned by more than three indicators in an AND relationship, enter the letters AND in positions 14 through 16 of the following line and specify the additional indicators in positions 23 through 31 on that line.

For an AND relationship, fetch overflow (position 16) can only be specified on the first line. Positions 17 through 22 (spacing and skipping) must be blank for all AND lines.

An overflow indicator must be defined on the file description specifications (positions 33 and 34) before it can be used as a conditioning indicator. If a line is to be conditioned as an overflow line, the overflow indicator must appear on the main specification line or on the OR line. If an overflow indicator is used on an AND line, the line is *not* treated as an overflow line, but the overflow indicator is checked before the line is written. In this case, the overflow indicator is treated like any other output indicator.

If the output record is to be written when any one of two or more sets of conditions exist (an OR relationship), enter the letters OR in positions 14 and 15 of the following specification line, and specify the additional OR indicators on that line.

When an OR line is specified for a printer file, the skip and space entries (positions 17 through 22) can all be blank, in which case the space and skip entries of the preceding line are used. If they differ from the preceding line, enter space and skip entries on the OR line. If fetch overflow (position 16) is used, it must be specified on each OR line.

Positions 32-37 (EXCPT Name)

When the record type is an exception record (indicated by an E in position 15), a name can be placed in these positions of the record line. The EXCPT operation can specify the name assigned to a group of the records to be output. This name is called an EXCPT name. An EXCPT name must follow the rules for using symbolic names. A group of any number of output records can use the same EXCPT name, and the records do not have to be consecutive records.

When the EXCPT operation is specified without an EXCPT name, only those exception records without an EXCPT name are checked and written if the conditioning indicators are satisfied.

When the EXCPT operation specifies an EXCPT name, only the exception records with that name are checked and written if the conditioning indicators are satisfied.

The EXCPT name is specified on the main record line and applies to all AND/OR lines.

If an exception record with an EXCPT name is conditioned by an overflow indicator, the record is written only during the overflow portion of the RPG cycle or during fetch overflow. The record is not written at the time the EXCPT operation is processed.

Field Description and Control Entries

Entries in positions 23 through 70 determine under what conditions and in what format fields of a record are to be written.

Each field is described on a separate line. No entries are permitted in positions 7 through 22 of a field description line. Field description and control information for a field begins on the line following the record identification line.

Positions 23-31 (Output Indicators)

Indicators specified on the field description lines determine whether a field is to be included in the output record. The same types of indicators can be used to control fields as are used to control records (see "Positions 23-31 (Output Indicators)" on page 7-13) under "Record Identification and Control Entries" on page 7-9 earlier in this chapter). Indicators used to condition field descriptions lines cannot be specified in an AND/OR relationship. Conditioning indicators cannot be specified on format name specifications for program described WORKSTN files.

Positions 32-37 (Field Name)

In positions 32 through 37, use one of the following entries to specify each field that is to be written out:

- A field name
- Blanks if constants are specified in positions 45 through 70
- A table name, array name, or array element
- The RPG/400 reserved words PAGE, PAGE1 through PAGE7, *PLACE, UDATE, UDAY, UMONTH, UYEAR, *IN, *INxx, or *IN,xx
- A data structure name or data structure subfield name.

Field Names, Blanks, Tables and Arrays

The field names used must be defined in the program. Do not enter a field name if a constant is used in positions 45 through 70. If a field name is entered in positions 32 through 37, positions 7 through 22 must be blank.

Fields can be specified in any order because the sequence in which they appear on the output records is determined by the entry in positions 40 through 43. If fields overlap, the last field specified is the only field completely written.

When a nonindexed array name is specified, the entire array is written. An array name with a constant index or variable index causes one element to be written. When a table name is specified, the element last found in a LOKUP operation is written. The first element of a table is written if no successful LOKUP operation was done.

The conditions for a field and the conditions for the record it is contained in must be satisfied before the field is written out.

PAGE, PAGE1-PAGE7

To use automatic page numbering, code PAGE in positions 32 through 35 as the name of the output field. Indicators specified in positions 23 though 31 do not condition the field, but rather control the resetting of the PAGE field. When indicators are specified and their conditions are met, the PAGE field is set to zero and incremented by 1 before output; when the conditions are not met, the PAGE field is incremented by 1 and then output. If page numbers are needed for several output files (or for different numbering within one file), the entries PAGE1 through PAGE7 can be used. The PAGE fields are automatically zero-suppressed by the Z edit code.

For more information on the PAGE reserved words, see Chapter 10, "RPG Words with Special Functions."

*PLACE

*PLACE is an RPG/400 reserved word that is used to repeat data in an output record. Fields or constants that have been specified on previous specification lines can be repeated in the output record without having the field and end positions named on a new specification line. When *PLACE is coded in positions 32 through 37, all data between the first position and the highest end position previously specified for a field in that output record is repeated until the end position specified in the output record on the *PLACE specification line is reached. The end position specified on the *PLACE specification line must be at least twice the highest end position of the group of fields to be duplicated. *PLACE can be used with any type of output. Blank after (position 39), editing (positions 38, 45 through 70), data format (position 44), and relative end positions cannot be used with *PLACE.

UDATE, UDAY, UMONTH, UYEAR

The user date reserved words allow the programmer to supply a date for the program at run time. For more information on the user date reserved words, see Chapter 10, "RPG Words with Special Functions."

*IN, *INxx, *IN,xx

The reserved words *IN, *INxx and *IN,xx allow the programmer to refer to and manipulate RPG/400 indicators as data.

Position 38 (Edit Codes)

Position 38 is used to specify edit codes that suppress leading zeros in a numeric field or to punctuate a numeric field without using an edit word. Allowable entries are 1 through 9, A through D, J through M, X, Y, Z, and blank.

Edit codes 5 through 9 are user-defined edit codes and are defined externally to RPG/400 by an OS/400 function. The edit code used by RPG/400 is determined at compilation time. Subsequent changes to a user-defined edit code will not affect the editing by RPG/400 unless the program is recompiled.

Position 39 (Blank After)

Entry	Explanation
Blank	The field is not reset.
B	The field specified in positions 32 through 37 is reset to blank or zero after the output operation is complete.

Position 39 is used to reset a numeric field to zeros or a character field to blanks. If the field is conditioned by indicators in positions 23 through 31, the blank after is also conditioned. This position must be blank for look-ahead, UDATE, UMONTH, UDAY, UYEAR, *PLACE, and constants.

Resetting fields to zeros is useful when totals are accumulated and written for each control group in a program. After the total is accumulated and written for one

control group, the total field can be reset to zeros before accumulation begins on the total for the next control group.

If blank after (position 39) is specified for a field to be written more than once, the B should be entered on the last line specifying output for that field, or else the field named may be set to blanks or zeros before it is written again.

Positions 40-43 (End Position in Output Record)

Entry	Explanation
1-n	End position in output record
K1-K8	Length of format name for WORKSTN file.

Positions 40 through 43 define the end position of a field or constant on the output record, or define the length of the data description specifications record format name for a program described WORKSTN file.

The K identifies the entry as a length rather than an end position, and the number following the K indicates the length of the record format name. For example, if the format name is 'CUSPMT'., the entry in positions 42 and 43 is K6. Leading zeros are permitted following the K, and the entry must be right-adjusted.

Valid entries for end positions are blanks, ±nnn, and nnnn. All entries in these positions must end in position 43. Enter the position of the rightmost character of the field or constant. The end position must not exceed the record length for the file.

If an entire array is to be written, enter the end position of the last element in the array in positions 40 through 43. If the array is to be edited, be careful when specifying the end position to allow enough positions to write all edited elements. Each element is edited according to the edit code or edit word.

The ±nnn entry specifies the placement of the field or constant relative to the end position of the previous field. The sign must be in position 40. The number (nnn) must be right-adjusted, but leading zeros are not required. To calculate the end position, use the formula:

EP = PEP ±nnn + FL

EP is the calculated end position. PEP is the previous end position. For the first field specification in the record, PEP is equal to zero. FL is the length of the field after editing, or the length of the constant specified in this specification. The use of +nnn is equivalent to placing nnn positions between the fields. A -nnn causes an overlap of the fields by nnn positions. For example, if the previous end position (PEP) is 6, the number of positions to be placed between the fields (nnn) is 5, and the field length (FL) is 10, the end position (EP) equals 21.

When *PLACE is used, an actual end position must be specified; it cannot be blank or a displacement.

An entry of blank is treated as an entry of +000. No positions separate the fields.

Position 44 (Packed/Binary, Sign Left/Right)

Entry	Explanation
Blank	The field is to be written in zoned decimal numeric or character format; numeric data is edited, or a constant is specified on this line.
P	The field is to be written in packed decimal format.
B	The field is to be written in binary format.
L	The numeric output field is to have a preceding (left) plus or minus sign.
R	The numeric output field is to have a following (right) plus or minus sign.

This position must be blank if editing is specified.

The entry in position 44 specifies the format of the data in the records in the file. This entry has no effect on the format used for internal processing of the output field in the program.

Positions 45-70 (Constant or Edit Word)

Positions 45 through 70 are used to specify a constant, an edit word, or a format name for a program described file.

Constants

Constants consist of character data (literals) that does not change from one processing of the program to the next. A constant is the actual data used in the output record rather than a name representing the location of the data.

A constant of up to 24 characters can be placed in positions 45 through 70. The constant must begin in position 46 (apostrophe in position 45), and it must end with an apostrophe even if it contains only numeric characters. Any apostrophe used within the constant must be entered twice; however, only one apostrophe appears when the constant is written out. The field name (positions 32 through 37) must be blank.

Edit Words

An edit word specifies the punctuation of numeric fields, including the printing of dollar signs, commas, periods, and sign status.

Format Name

The name of the data description specifications record format that is used by a program described WORKSTN file must be specified in positions 45 through 54. One format name is required for each output record for the WORKSTN file; specifying more than one format name per record is not allowed. Conditioning indicators cannot be specified on format name specifications for program described WORKSTN files. The format name must be enclosed in apostrophes. You must also enter Kn in positions 40 through 43, where n is the length of the format name. For example, if the format name is 'CUSPMT'., enter K6 in positions 42 and 43.

Positions 71-74

Positions 71 through 74 must be blank.

Positions 75-80

Positions 75 through 80 can be used for comments, or left blank.

Externally Described Files

Position 6 (Form Type)

An O must appear in position 6 to identify this line as an output specifications statement.

Record Identification and Control Entries

Output specifications for an externally described file are optional. Entries in positions 7 through 37 of the record identification line identify the record format and determine under what conditions the records are to be written.

Positions 7-14 (Record Name)

Entry	Explanation
A valid record format name	A record format name must be specified for an externally described file.

Positions 14-16 (AND/OR)

Entry	Explanation
AND or OR	AND/OR indicates a relationship between lines of output indicators. AND/OR lines are valid for output records, but not for fields.

See "Positions 14-16 (AND/OR)" on page 7-9 for more information.

Position 15 (Type)

Entry	Explanation
H or D	Detail records
T	Total records
E	Exception records.

Position 15 indicates the type of record to be written. See "Position 15 (Type)" on page 7-10 under "Program Described Files" on page 7-8 earlier in this chapter for more information.

Position 16 (Release)

Entry	Explanation
R	Release a device after output.

See "Position 16 (Fetch Overflow/Release)" on page 7-11 under "Program Described Files" on page 7-8 earlier in this chapter for more information.

Positions 16-18 (ADD/DEL)

Entry	Explanation
ADD	Add a record to a file.
DEL	Delete an existing record from the file.

For more information on ADD/DEL, see "Positions 16-18 (ADD/DEL)" on page 7-10.

Positions 16-22 (Fetch Overflow/Space/Skip)

The fetch overflow, space, and skip entries for an externally described file must be blank. Space and skip entries for an externally described printer file are specified in the data description specifications.

Positions 23-31 (Output Indicators)

Output indicators for externally described files are specified in the same way as those for program described files. The overflow indicators OA-OG, OV are not valid for externally described files. For more information on output indicators, see "Positions 23-31 (Output Indicators)" on page 7-13 under "Program Described Files" on page 7-8 earlier in this chapter.

Positions 32-37 (EXCPT Name)

An EXCPT name can be specified in these positions for an exception record line. See "Positions 32-37 (EXCPT Name)" on page 7-14 under "Program Described Files" on page 7-8 earlier in this chapter for more information.

Field Description and Control Entries

For externally described files, the only valid field description and control entries are conditioning indicators (positions 23 through 31), field name (positions 32 through 37), and blank after (position 39).

Positions 23-31 (Output Indicators)

Indicators specified on the field description lines determine whether a field is to be included in the output record. The same types of indicators can be used to control fields as are used to control records. See "Positions 23-31 (Output Indicators)" on page 7-13 under "Program Described Files" on page 7-8 earlier in this chapter for more information.

Positions 32-37 (Field Name)

Entry	Explanation
Valid field name	A field name specified for an externally described file must be present in the external description unless the external name was renamed for the program.
*ALL	Specifies the inclusion of all the fields in the record.

For externally described files, only the fields specified are placed in the output record. *ALL can be specified to include all the fields in the record. If *ALL is specified, no other field description lines can be specified for that record. In particular, you cannot specify a B (blank after) in position 39.

For an update record, only those fields specified in the output field specifications and meeting the conditions specified by the output indicators are placed in the output record to be rewritten. The values that were read are used to rewrite all other fields.

For the creation of a new record, the fields specified are placed in the output record. Those fields not specified or not meeting the conditions specified by the output indicators are written as zeros or blanks, depending on the data format specified in the external description.

Position 38 (Edit Codes)

Position 38 must be blank because edit codes for externally described files are specified in the data description specifications.

Position 39 (Blank After)

Entry	Explanation
Blank	The field is not reset.
B	The field specified in positions 32 through 37 is reset to blanks or zeros after the output operation is complete. Do not specify *ALL in positions 32 through 37.

For more information on blank after, see "Position 39 (Blank After)" on page 7-16.

Positions 40-43 (End Position in Output Record)

These positions must be blank because the field is placed in the output record in the positions specified by the data description specifications.

Position 44 (Packed/Binary, Sign Left/Right)

This position must be blank because the data format is defined in the data description specifications.

Positions 45-70 (Constant or Edit Word)

These positions must be blank because editing is specified in the data description specifications. If a constant is to be placed in an externally described file, it must be done by a calculation operation.

Positions 71-74

Positions 71 through 74 must be blank.

Positions 75-80

Positions 75 through 80 can be used for comments, or left blank.

Chapter 8. Auto Report Feature

The RPG/400 auto report function is a program that operates before the RPG/400 compiler. Auto report on the AS/400 System is intended to be used for conversion of existing auto report programs. The use of auto report with AS/400 RPG/400 enhancements such as externally described files or the DEFN operation code may yield undesirable results.

Auto report uses the following specifications to generate a complete RPG/400 source program that is ready to be compiled:

- Auto report option specifications
- *AUTO page headings and *AUTO output specifications in the source program
- Standard RPG/400 specifications in the source program
- Auto report /COPY statements in the source program, with or without modifier statements
- Standard RPG/400 specifications, including arrays and tables, and *AUTO specifications that are copied from a source file member by the auto report copy function (compile-time arrays and tables are not changed by auto report; they remain in the standard array/table record format).

Format of the Auto Report Generated Specifications

The positions under the page/line heading on the compiler listing indicate the following.

Position	Contents
1-4	Sequence number of the specification. This number starts as 0001 on the RPG/400 control specification and is incremented by 0001 on each specification that follows. If more than 9999 specifications are present in the program, the sequence is restarted at 0000.
5	Code that identifies the specification as follows:
	Blank Standard RPG/400 specification present in the auto report program.
	C Specification copied from a source file member.
	M Specification copied from a source file member and modified.
	E Specification generated by auto report.

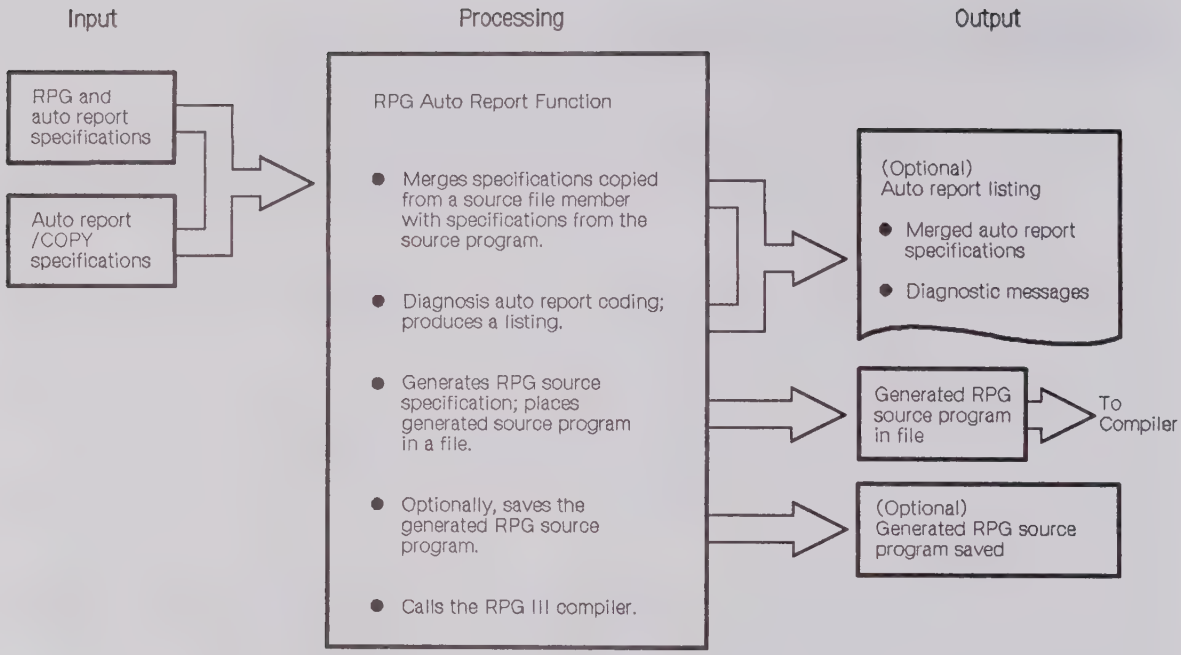


Figure 8-1. Operations of Auto Report Function

Order of Auto Report Generated Specifications

Auto report generates the specifications in the order required by the RPG/400 compiler. When specifications are included through a /COPY statement, those specifications are initially placed immediately after the /COPY statement. After all specifications are copied and before auto report generates RPG/400 specifications from the H-*AUTO and D/T-*AUTO specifications, the entire auto report source program is sorted into the following order:

1. Control specification
2. File description specifications
3. Extension specifications
4. Line counter specifications
5. Input record specifications
6. Data structure specifications
7. Calculation specifications (in the order: detail, total, and subroutines)
8. Output specifications
9. Arrays and tables loaded at compilation time, which must be placed last among the input statements to auto report.

Calculation Specifications

Generated RPG/400 calculation specifications are placed in the following order by auto report:

1. Detail calculations specified by the programmer
2. EXSR statement for the generated subroutine
3. Total calculations generated by auto report, grouped in order by level (all L0 calculations, then all L1 calculations, and so on)
4. Total calculations specified by the programmer
5. Subroutines specified by the programmer
6. Generated RPG/400 subroutine that accumulates the lowest level total.

Output Specifications

Output heading specifications generated for H-*AUTO specifications appear in the same order they are coded on the output specifications relative to other RPG/400 and *AUTO output specifications for the file.

Normally, RPG/400 output specifications generated from a D/T-*AUTO specification are in the following order:

1. Heading specifications generated for column headings
2. Detail specifications
3. Total specifications, with the lowest level first and LR last.

This group of specifications is placed in the same relative position in the program as the original D/T-*AUTO specification. All other RPG/400 output specifications remain in their original order.

If, however, the programmer specifies a normal RPG/400 total output specification conditioned by a positive control level indicator (no N in position 23) in positions 24 and 25 for the file that has a D/T-*AUTO specification, all output specifications in the program are sorted into the following format:

1. All heading, detail, and exception output specifications remain in the same order as they are in the generated RPG/400 source program. Total specifications that are not conditioned by a positive control level indicator in positions 24 and 25 remain as they were in the program.
2. Total specifications, which are conditioned by a positive control level indicator in positions 24 and 25, are sorted into ascending order according to the control level indicator in positions 24 and 25, with LR last.

Comment Statements

Comment statements (identified by an asterisk in position 7) are allowed among the statements read by auto report. However, because the sorting of RPG/400 specifications is determined by the contents of position 6, comments may not occur in the expected order. To ensure that comments remain with the correct specification, place them after that specification and put the same entry in position 6.

Restriction

The order of arrays and tables is not altered when the source specifications are sorted. Therefore, when arrays and tables are included from a source file member, they may not occur in the correct order after the sort. For example, if a file translation or alternate collating sequence table is present in the auto report source specifications, then any compile-time arrays or tables included from a source file member are out of order. That is, the included arrays or tables are placed ahead of the file translation table. Compile-time arrays and tables must be loaded in the following order:

1. File translation specifications
2. Alternate collating sequence specifications
3. Compile-time arrays and tables in the order described on the extension specifications form.

A solution to this restriction is to place the file translation and alternate collating sequence tables in a source file member and copy them from the source file member before any other compile-time arrays and tables are copied. This procedure ensures that the file translation and alternate collating sequence tables are the first compile-time tables in the generated RPG/400 source program.

Options Specifications

All the options for the auto report program can be specified in the CL command CRTRPTPGM (Create Auto Report Program). Any options specified in the CRTRPTPGM command override the options specified on the auto report option specifications. Only the options date suppress and *suppress can be specified on the auto report option specifications (see Figure 8-2).

The auto report option specification is not required in the auto report program. If present, it must appear as the first specification in the program. If it is not present, auto report uses the options that are specified in the CRTRPTPGM command. Option specifications cannot be contained in a source file member that is copied by a /COPY statement.

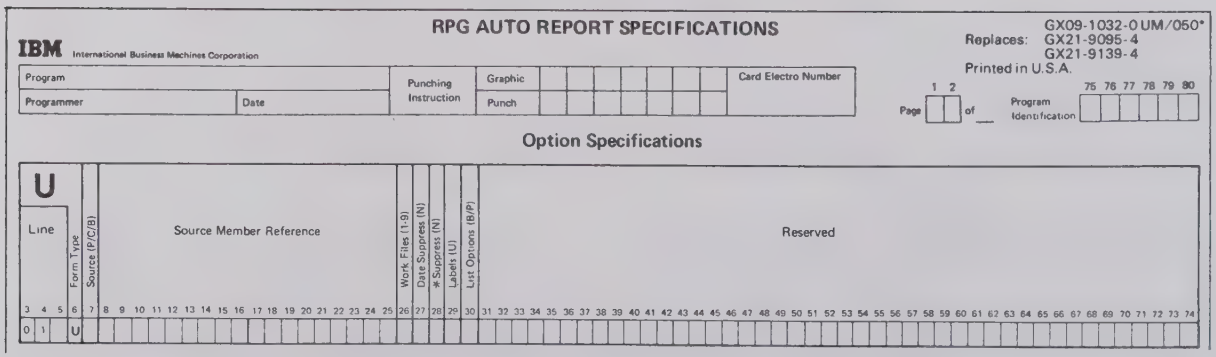


Figure 8-2. RPG/400 Auto Report Specifications

If the RPG/400 control specification (H in position 6) is not present either in the auto report source program or in a copied source file member auto report generates a control specification with blank entries. See Chapter 1, "Control Specifications" for the meaning of blank entries for the control specification.

The following positions on the auto report specifications form are used in the same way as corresponding positions on standard RPG/400 specifications forms.

- Positions 1-2 (page)
- Positions 3-5 (line)
- Positions 75-80 (program identification).

Position 6 (Form Type)

Enter a U in position 6 to identify this line as an auto report option specification.

Positions 7-26

Positions 7 through 26 are not used. Leave them blank. Entries in positions 7 through 26 are ignored.

Position 27 (Date Suppress)

Entry	Explanation
Blank	Page number and date are included on the first *AUTO page heading line.
N	Date and page number on the first *AUTO page heading line are suppressed.

To suppress the generated date and page number from printing on the first *AUTO heading line, enter an N in position 27. When these fields are suppressed, the page title and any other fields specified can occupy the entire line. See "*AUTO Page Heading Specifications" on page 8-7 for further information on the generated date and page numbers.

Date and page number suppression can also be specified by the *NODATE option of the RPTOPT parameter of the CRTRPTPGM command. The command option overrides the entry in position 27.

Position 28 (*Suppress)

Entry	Explanation
Blank	Asterisks are generated for total output lines.
N	Asterisk indication is suppressed from generated total output lines.

To suppress asterisks from printing beside generated totals, enter an N in position 28. See "*AUTO Output Specifications" on page 8-10 for rules used in generating asterisk indication.

Asterisk suppression can also be specified by the *NOAST option of the RPTOPT parameter in the CRTRPTPGM command. The command option overrides the entry in position 28.

Positions 29-74

Positions 29 through 74 must be blank.

*AUTO Specifications

The *AUTO page heading function and the *AUTO output function provide simplified methods of describing printed output. These functions of auto report are requested when the characters *AUTO are present in positions 32 through 36 of a record description specification on the standard RPG/400 output specifications form. *AUTO can be entered on a heading, detail, or total specification (H, D, or T in position 15), but not on an exception output specification (E in position 15). Use *AUTO with only one printer file in the program.

Standard RPG/400 output specifications are divided into two general types (see Figure 8-3).

- *Record description specifications* (positions 7 through 31) describe when and where the output line is to be printed. One record description specification is required for each different type of line to be printed. Only the first record description for a file need contain a file name in positions 7 through 14.
- *Field description specifications* (positions 23 through 74) following a record description specification tell when, where, and how each item of data (field or literal) is to be printed on the output record. There can be several field description specifications following a record description specification.

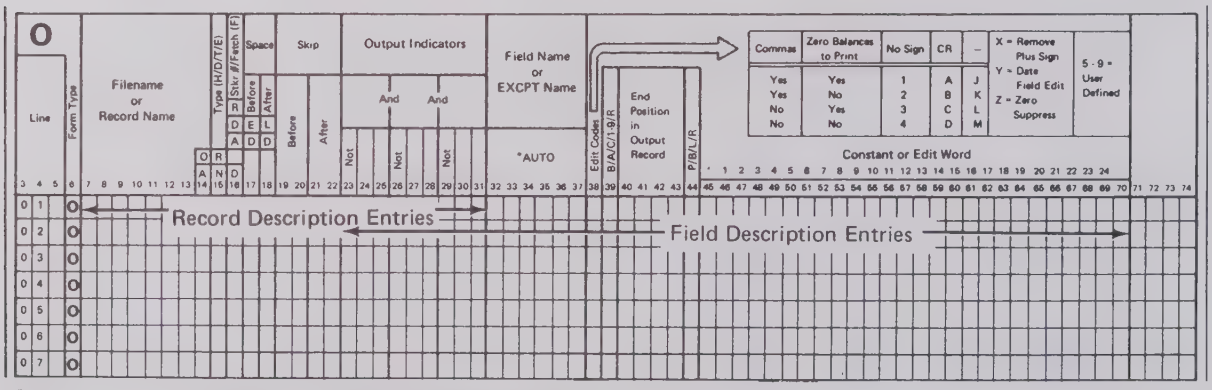


Figure 8-3. Two Categories of Output Specifications

Auto report page headings and auto report output specifications are also divided into the two general categories: record description specifications and field description specifications. However, the use of entries on these specifications is different from the use of entries for standard RPG/400 specifications.

The following output specifications are not changed when they are used with *AUTO.

- Positions 1-2 (page)
- Positions 3-5 (line)
- Position 6 (form type)
- Positions 75-80 (program identification).

Positions 71 through 74 must always be blank on auto report output specifications.

*AUTO Page Heading Specifications

The *AUTO page heading specifications provide an easy way to produce a page heading at the top of every page of a printed report (see Figure 8-4 on page 8-8). Up to five *AUTO page heading specifications can be used for a multiple-line page heading. If both standard RPG/400 heading lines and *AUTO page headings are specified in combination for a file, they are printed in the order specified by the output specifications. The *AUTO page headings can be specified for only one printer file per program.

The heading line generated by the first *AUTO page heading (H-*AUTO) specification contains a date and page number. The first heading line can also contain a title (See "Field Description Specifications" on page 8-9 for information on entering a title.) The generated date is left-adjusted and prints with slashes as mm/dd/yy unless the format is altered by the RPG/400 date or inverted print option (positions 19 through 21 of the control specification). The generated page number is right-adjusted and is preceded by the word PAGE. The page number field is four digits long and is zero-suppressed. Auto report uses one of the unused PAGE fields (PAGE, PAGE1 through PAGE7) for page numbering. If all PAGE fields are used in the program, auto report does not number pages. To suppress the date and page number on the first heading line, enter an N in position 27 of the auto report option specifications, or use the *NODATE option of the RPTOPT parameter in the CRTRPTPGM command.

Record Description Specifications

Each H-*AUTO record description defines a separate heading line. The record description entries allow the programmer to specify spacing and skipping and the conditions under which the line is printed.

Positions 7-14 (File Name)

Enter the valid file name of the printer file on the first line that defines an output record for the file.

Position 15 (Type)

Enter an H in position 15 on each record description specification line that defines a page heading line. The H and the entry *AUTO in positions 32 through 36 define this as an H-*AUTO heading specification (see Figure 8-4 on page 8-8). Up to five H-*AUTO specifications are allowed.

Position 16

Position 16 must be blank.

Positions 17-22 (Spacing and Skipping)

Enter spacing and skipping values in these positions according to normal RPG/400 rules.

Positions 23-31 (Output Indicators)

On the first H-*AUTO specification, either leave positions 23 through 31 blank or enter output indicators according to the rules given under "Positions 23-31 (Output Indicators)" on page 7-13.

If these positions are blank, auto report causes the corresponding output line to be printed at first page (1P) time in the program cycle and when overflow occurs. Thus, the heading is printed at the top of each page of the printed report. Indicators can be assigned to subsequent H-*AUTO specifications. If positions 23 through 31 are blank on any H-*AUTO specification after the first, that specification is assigned the same indicators as the first.

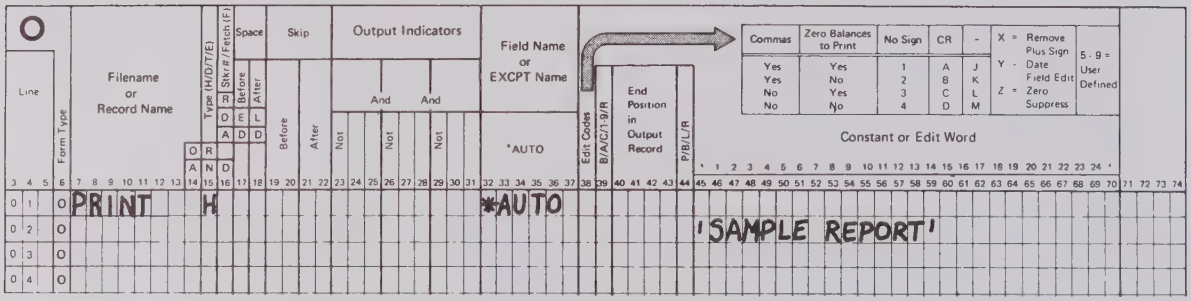


Figure 8-4. Specifications and Results – *AUTO Heading Lines

If an overflow indicator is specified on the file description specifications for the printer file, that indicator conditions the generated heading specifications. Otherwise, auto report defines an unused overflow indicator for the printer file and conditions the line with that indicator.

AND and OR lines can be used with H-*AUTO output indicators if an output indicator is used with the first specification. Standard RPG/400 rules for AND and OR lines apply.

Positions 32-37 (*AUTO)

Enter *AUTO in positions 32 through 36. This entry and an H in position 15 of the output specifications (see Figure 8-4) indicate that this is an auto report heading line.

Positions 38-70

Positions 38 through 70 are not used on the record description line. Leave them blank.

Field Description Specifications

Each H-*AUTO record description specification can be followed by one or more field description specifications. The field description specifications specify the title to be printed on the heading line and describe any other fields and literals to be printed on the line.

Positions 7-31

Positions 7 through 31 are not used on field description specifications. Leave them blank. Output indicators in positions 23 through 31 cannot be used to condition a field on an H-*AUTO specification.

Positions 32-37 (Field Name)

Entry	Explanation
Blank	A constant (enclosed in apostrophes) must be entered in positions 45 through 70. The constant is printed on the heading line.
Field name	Field defined in the program is printed on the heading line.
Table name	A table element is printed on the heading line.
Indexed array name	An array element is printed on the heading line.

Use positions 32 through 37 to enter a field name, table name, or indexed array name defined elsewhere in the program that is to print on the heading line. If a name is entered, an edit word, not a constant, can be entered in positions 45 through 70. A constant must be entered in positions 45 through 70 if positions 32 through 37 are blank.

If output indicators (positions 23 through 31) are left blank on the record description specification, auto report conditions all fields and array/table elements included on the heading line with N1P in positions 23 through 25. Therefore, the field or array/table element does not print on the first page. (If printed on the first page, the field may not contain meaningful data because the first record is not read.) N1P is not generated for the following RPG/400 reserved words: PAGE, PAGE1 through PAGE7, UDATE, UDAY, UMONTH, UYEAR.

Position 38 (Edit Codes)

An edit code can be entered in position 38 if a numeric field, numeric array element, or numeric table is named in positions 32 through 37. If an edit code is used, positions 45 through 70 must be blank unless asterisk protection or a floating currency symbol is specified. If position 38 is blank, no editing is done by auto report unless an edit word is used.

Position 39 (Blank After)

Enter a B in position 39 to reset a numeric field to zeros after it is printed or to reset a character field to blanks after it is printed on the heading line.

Positions 40-44

Positions 40 through 44 are not used with *AUTO heading specifications. Leave them blank.

Positions 45-70 (Constant or Edit Word)

Entry	Explanation
Blank	Positions 32 through 37 contain the name of a field that either is not edited or is edited by an edit code.
Constant	Title or other constant (enclosed in apostrophes) that is to appear on the printed line.
Edit word	The edit pattern used to edit the numeric field named in positions 32 through 37 of the same field description line.

Use positions 45 through 70 to specify the title and other information that is to appear on the output line and to edit numeric fields that are to appear on the line. No end positions can be specified.

*AUTO Output Specifications

Detail reports (where a line is printed for each individual record that is read) and group printed reports (where only totals are printed) can be specified by the *AUTO output function alone or in combination with standard RPG/400 specifications. The *AUTO output function generates totals and formats columns and column headings.

A single detail or total *AUTO record description (D/T-*AUTO) specification and its associated field description specifications can specify:

- Up to three lines of column headings to appear above a field
- Accumulation of several levels of totals, including a final total (known as *total rolling*)
- Generation by auto report of end positions for column headings and fields
- Generation by auto report of the K edit code for numeric fields
- Fields or constants to be printed next to generated totals.

Four types of field description specifications can be associated with the *AUTO record description specification. The four types are distinguished by entries in position 39. The remaining entries on a field description specification have different meanings depending on the entry in position 39.

The valid entries in position 39 of the field description specifications and their meanings are:

- *Blank or B:* Indicates the associated field or constant appears on the detail line.

- *A:* Indicates the associated numeric field is printed on the detail line and accumulated. A total is printed for each control level defined in positions 59 and 60 of the input specifications for the program. A final total is also printed (when the LR indicator is on).
- *C:* Indicates the associated constant is printed on the second or third line of column headings.
- *1, 2, 3, 4, 5, 6, 7, 8, 9, R:* Indicates the associated field or constant appears on the total line generated for the respective control level indicator (L1 through L9, LR).

Record Description Specifications

An auto report record description specification must contain the entry *AUTO in positions 32 through 36. *AUTO can appear only on a record description specification. This entry indicates that the record description and the following field descriptions are redefined according to their use by auto report.

Positions 7-14 (Filename)

Enter the valid file name of the printer file on which the report is to be printed. This must be the same file named on H-*AUTO specifications, if any.

Position 15 (Type)

Entry	Explanation
D	The auto report specifications describe a report containing detail lines.
T	The auto report specifications describe a report containing total lines, but no detail lines (group printed report).

Enter a D in position 15 and *AUTO in positions 32 through 36 for auto report to generate a report that contains detail lines. The field description specifications associated with the D-*AUTO record description specify:

- Fields to appear on the detail line
- Column headings
- Total rolling
- Constants to appear on total lines.

Enter a T in position 15 and *AUTO in positions 32 through 36 for auto report to generate a group printed report.

Only one detail or one total *AUTO (D/T-*AUTO) record description specification can be used in a program.

Position 16 (Fetch Overflow)

Enter an F in position 16 to specify fetch overflow.

When used with the *AUTO output function, fetch overflow applies only to the detail line. If group printing is specified (T in position 15), fetch overflow applies to the lowest level total line to be printed.

Positions 17-22 (Spacing and Skipping)

Enter spacing and skipping values in positions 17 through 22 according to the standard RPG/400 rules. Entries specified apply only to the detail line generated by a D-*AUTO specification or to the first total line generated by a T-*AUTO specification.

Leave positions 17 through 22 blank to single space after each detail line printed or, if group printing is specified, after the first total line printed.

Positions 23-31 (Output Indicators)

Enter any valid output indicators in positions 23 through 31 to condition the detail or group print line generated by this *AUTO specification. If these positions are left blank on a D-*AUTO specification, the generated detail line is conditioned by N1P. Therefore, it is not printed at first page (1P) time in the RPG/400 program cycle. If these positions are left blank for a T-*AUTO specification, the first generated total line is conditioned by the lowest control level indicator defined in the program.

AND and OR can be used with *AUTO output indicators if an output indicator is specified on the first record description specification. Standard RPG/400 rules for AND and OR lines apply.

Indicators specified in positions 23 through 31 of the record description specification (and its associated AND/OR lines) apply only to the detail line generated by a D-*AUTO specification or the group print line (lowest level total specification) generated by a T-*AUTO specification.

If column headings are specified in the field description specifications that follow this *AUTO record description, they are conditioned by one of the following:

- The same indicators that are specified for the first H-*AUTO specification.
- The first page (1P) indicator in an OR relationship with the overflow indicator specified for the file on the file description specifications. If no overflow indicator is specified, auto report defines an unused overflow indicator and uses it to condition the lines.

Restriction: If N1P is specified on a D-*AUTO record description specification that is followed by field description specifications for totaling fields (A in position 39), the calculations generated for the totaling fields are also conditioned by N1P. This causes a diagnostic in the RPG/400 compiler.

Positions 32-37 (*AUTO)

To indicate that this is an auto report specification, enter *AUTO in positions 32 through 36 on the record description line. Position 15 must contain a D or a T to indicate a detail or total *AUTO specification. Only one D/T-*AUTO specification can be used in a program.

Positions 38-70

Positions 38 through 70 are not used on a D/T-*AUTO record description specification. Leave them blank.

Field Description (Blank or B in Position 39)

D-*AUTO and T-*AUTO field description specifications containing a blank or B in position 39 describe:

- A character field such as an item description
- A numeric field that is not totaled
- A constant
- A field with a literal to be used as a column heading (see Figure 8-5 on page 8-14).

A field named on the line (or a constant when no field is named) following a D-*AUTO record description specification is printed only on the detail report line. If the field (or constant when no field is named) on the line follows a T-*AUTO record description, it appears only on the first total line generated.

Positions 7-22

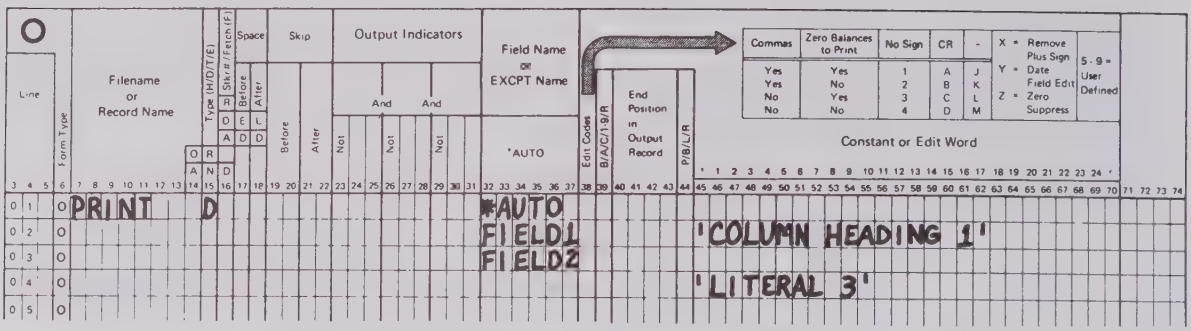
Positions 7 through 22 are not used on the field description lines. Leave them blank.

Positions 23-31 (Output Indicators)

Enter any valid RPG/400 output indicators in positions 23 through 31 or leave them blank. If these positions are left blank, the field (or constant when no field is named on the line) is printed on each detail line conditioned by the indicator. When group printing is specified (T-*AUTO specification), the field (or constant when no field is named on the line) is printed each time the lowest level total line is printed. If a column heading is specified in positions 45 through 70 to appear over a field named in positions 32 through 37, the column heading is not affected by output indicators entered in positions 23 through 31.

Positions 32-37 (Field Name)

Enter a field name, data structure name, indexed array name, table name, or blanks in positions 32 through 37. If positions 32 through 37 are blank, a constant must be entered in positions 45 through 70 of the same field description specification. If a field name, data structure name, indexed array name, or table name is entered, the value of the field or element is printed on the detail line (or on the first total line if group printing is specified).



As a result of these specifications, FIELD1 prints on each detail line under the heading COLUMN HEADING 1. FIELD2 and LITERAL 3 print on each detail line without a column heading.

Figure 8-5. Auto Report Field Description Specifications (Blank in Position 39)

Position 38 (Edit Codes)

Enter a valid RPG/400 edit code in position 38 if positions 32 through 37 contain the name of a numeric field, a numeric array element, or a numeric table. This position must be blank for character fields, data structures, and array/table elements, and for literals. If position 38 is left blank on a field description line for a numeric field or array/table element, the auto report program provides a K edit code. The K edit code causes a numeric field or element to be printed with commas and a decimal point, such as 3,489.13. It also causes zero suppression; zero balances are not printed, and negative balances are printed with a minus sign on the right.

Position 39 (Blank After)

Entry	Explanation
Blank	Field is not to be reset to zeros or blank after printing.
B	Numeric field is reset to zeros after it is printed; character field is reset to blanks.

Enter a B in position 39 to reset character fields or data structures to blanks or to reset numeric fields to zeros after they are printed. Blank after cannot be used for constants. This entry applies only to the detail line (or the first total line if group printing is specified).

Positions 40-43 (End Position in Output Record)

Either leave positions 40 through 43 blank or enter the print position of the right-most character of the field (or constant if no field is named in positions 32 through 37) to be printed. If this position is blank, auto report generates end positions for fields, constants, and column headings.

Position 44

Position 44 is not used because packed and binary data cannot be specified. Leave this position blank.

Positions 45-70 (Constant)

Enter a constant or blanks in positions 45 through 70 if position 39 contains a blank. Constants are enclosed in apostrophes according to the standard RPG/400 rules for coding constants. If these positions are left blank, a field name, data structure name, indexed array name, or table name must be entered in positions 32 through 37. Column heading continuation lines can follow this field description line, but the first line of the printed column heading will be blank. See "Field Description (C in Position 39)" on page 8-19.

If a constant is entered in these positions along with a field name in positions 32 through 37, the constant is printed on the first column heading line over the field value. If a column heading is used, the length used to space the column on the report is the greater of the longest column heading length or the field length, adjusted for editing.

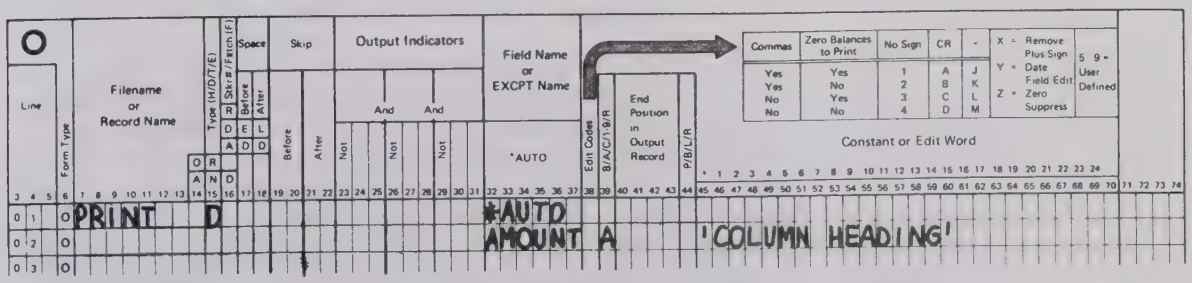
If a constant is entered in positions 45 through 70 and field name (positions 32 through 37) is blank, the constant is printed each time the detail report line is printed. In group printing, the constant is printed each time the first generated total line is printed.

Field Description (A in Position 39)

Enter an A in position 39 of a field description specification following a D/T-*AUTO specification to accumulate and print totals for the field named in positions 32 through 37 (see Figure 8-6). As many levels of totals are printed as are defined in the control level entry (positions 59 and 60) of the input specifications. A final total is also printed when the LR indicator is on. (This process is called *total rolling).*

If group printing is specified and a control level indicator higher than the lowest defined control level is specified in positions 23 through 31 on the record description specification, totals are generated for the indicator entered, all higher defined indicators, and LR.

The total output record generated by auto report if you have entered an A in position 39 of a field description specification is conditioned by the associated control level indicator defined in the input specifications. One total output record is generated for each control level indicator defined in the program.



The A in position 39 causes the AMOUNT field to be accumulated. Totals are printed for each control level and a final total is printed. A column heading is specified in positions 45 through 70.

Figure 8-6. Describing a Field to be Accumulated

Generated Total Fields

If an A is specified in position 39 of a detail or total *AUTO field description specification, auto report generates and names total fields to be used in accumulating the required levels of totals. Auto report generates the field names for the total fields based on the name in positions 32 through 37 of the A-type field description. Names are generated in the following way:

- If the specified field name has fewer than 6 characters, 1 character is added to the name to create a name for the total field. The added character is 1 through 9 or R, corresponding to the total indicators L1 through L9 and LR, respectively. For example, if ITEM is the specified field name and all nine control levels are defined, the generated field names are ITEM1, ITEM2, . . . ITEM9, and ITEMR.
- If the specified field name has 6 characters, the last character is replaced by one of the characters, 1 through 9, or R. For example, if AMOUNT is the specified field name and all nine control levels are defined, the generated field names are AMOUN1, AMOUN2, . . . AMOUN9, and AMOUNR.

Total fields are generated and named for all control level indicators defined in the program and for LR. For example, if L1 and L3 are assigned to control fields on the input specifications and the field QTY is specified, three total fields are generated and named by auto report: QTY1, QTY3, and QTYR. All total fields generated for the same level, such as QTY1 and AMOUN1, are printed on the same total line, and that line is conditioned by the corresponding control level indicator.

Generated total fields are two digits longer than the original field. For example, if the field QTY is defined with a length of three, QTY1, QTY3, and QTYR all have lengths of five. The number of decimal positions remains the same in the generated fields. You can redefine a field name previously defined in a program that is the same as a generated field name, giving that field whatever length and number of decimal positions you want. If you do this, the generated field is assigned the previously defined length and number of decimal positions (if the previous field is numeric).

Considerations

Generated field names can be used in RPG/400 specifications that are included in the program. The programmer must be aware, however, that the use of generated fields in this way can interfere with the automatic accumulation of totals done by auto report.

Field names ending in 1 through 9 or R should not be used in an auto report program that accumulates totals, because auto report generates total fields ending in those characters. This is especially important for 6-character field names because auto report forms total field names by replacing the last character with 1 through 9 or R. No field name can be used more than once with an A in position 39. Also, if a 5- or 6-character field name is specified with an A in position 39, a second 5- or 6-character field name in which the first 5 characters are identical cannot be specified with an A in position 39. For example, if the following four field names are specified with an A in position 39 in an auto report specification, all but the first are invalid:

Field	Status
FIELD	Valid.
FIELDX	Invalid because the first 5 characters duplicate the first 5 characters of the first field.
FIELDY	Invalid for the same reason as that for FIELDX.
FIELD	Invalid because it is a duplicate of the first field.

Positions 7-22

Positions 7 through 22 must remain blank on the field description lines.

Positions 23-31 (Output Indicators)

Enter any valid RPG/400 output indicators in positions 23 through 31 or leave them blank. If these positions are blank, the field described is printed on each detail line. If indicators are entered in positions 23 through 31, the field is printed only when the conditions represented by those indicators are met. Leave these positions blank for group printing.

If a column heading is specified in positions 45 through 70 to appear over a field named in positions 32 through 37, the column heading is not affected by output indicators entered in these columns. Also, output indicators specified when position 39 contains an A do not affect the generation of calculations for the field.

Output indicators specified on an A-type field description specification following a D-*AUTO specification condition the calculations generated for the field. If the AL-type field description follows a T-*AUTO specification, however, a specified indicator does not condition calculations generated for the field.

Positions 32-37 (Field Name)

If position 39 contains an A, the name of a numeric field that is to be accumulated must be entered in positions 32 through 37. These positions cannot identify an array, array element, or table. The field named is printed on each detail line of the report. If group printing is specified, the total field for the lowest control level indicator defined (L1, L2, . . . L9, LR, in that order) is printed on the generated total line. Totaling for any particular field by means of an A entry in position 39 can be specified only once in each program.

To generate calculation and output specifications that accumulate and print the various levels of totals required, auto report creates and names additional totaling fields. Names for the fields are constructed based on the field name specified in these positions according to a set of rules (see "Generated Total Fields" on page 8-16).

Position 38 (Edit Codes)

Enter an edit code in position 38 or leave it blank. If this position is blank, auto report generates a K edit code for the field named in positions 32 through 37. This causes the field to be edited with commas and a decimal point, such as 1,234,567.89. The field is also zero-suppressed. Zero balances are not printed; negative balances are printed with a minus sign on the right. The edit code specified, or the generated

K edit code, applies to all generated total fields as well as to the field named in positions 32 through 37.

Position 39

Enter an A in position 39 to indicate that totals are to be accumulated for the field named in positions 32 through 37 of this field description. A total is printed for every control level indicator defined in the input specifications and for the LR indicator. If position 39 contains an A, positions 32 through 37 must contain the name of a numeric field. Positions 45 through 70 can contain a constant to be used as the first line of a column heading.

If the lowest control level indicator used for a T-*AUTO specification is higher than the lowest control level indicator defined in the input specifications, auto report generates only the total lines corresponding to the lowest control level indicator used for the T-*AUTO specification, the higher defined control levels, and LR.

Resetting Total Fields to Zero: If position 39 contains an A, the auto report program generates a B (blank after) in position 39 of all the detail and total field description specifications generated from the field name specified. Thus, the value in the specified field and in any generated fields is reset to zero after the field value is printed.

If group printing is specified, auto report generates a calculation to reset the specified field to zero on each cycle. This prevents the same value from being accumulated more than once. An unconditioned total calculation operation (Z-ADD) sets the field value to zero. This calculation is the first total calculation in the generated RPG/400 source program.

Asterisk Indication: To indicate that a printed line is a generated total line, asterisks are printed on the line to the right of the highest end position generated from the D/T-*AUTO specification. One asterisk is printed to the right on the lowest level total line generated. One additional asterisk is printed on each higher level line, including the final total.

For example, if L1 and L3 are defined control level indicators in a program, one asterisk is printed to the right of the L1 line, two asterisks are printed on the L3 line, and three are printed on the LR line. As many as 10 asterisks are printed on the LR line if all nine control level indicators are defined in the program.

To suppress the generation of asterisks on total lines, enter an N in position 28 of the auto report option specifications, or use the *NOAST option of the RPTOPT parameter in the CRTRPTPGM command.

Positions 40-43 (End Position in Output Record)

Enter the print position of the rightmost character of the field to be printed, or leave these positions blank. If this entry is blank, auto report generates end positions for fields and column headings.

Position 44

Position 44 is not used with auto report because packed and binary data cannot be used. Leave these positions blank.

Positions 45-70 (Constant)

Either leave positions 45 through 70 blank or enter a constant. Do not enter an edit word; editing is accomplished by an edit code. If a constant is entered when position 39 contains an A, the constant becomes the first line of the column heading over the accumulated field.

If these positions are left blank, the first line of the column heading is blank, but column heading continuation lines can specify the second and third line of the column heading. See "Field Description (C in Position 39)" for information on how column heading and fields are centered and spaced by auto report.

Field Description (C in Position 39)

Enter a `C` in position 39 of the `*AUTO` field descriptions to specify a second and third column heading line. At times you may want more information in a column heading than can be contained on one line. Auto report enables you to specify the second and third lines of column headings by simply specifying the constants to appear on those lines. No additional heading output lines need be coded; no end position need be calculated. The special field description specification that allows you to do this is identified by a C in position 39 (see Figure 8-7 on page 8-20).

Positions 7-38

Positions 7 through 38 must be blank on a field description that has a C in position 39.

Position 39

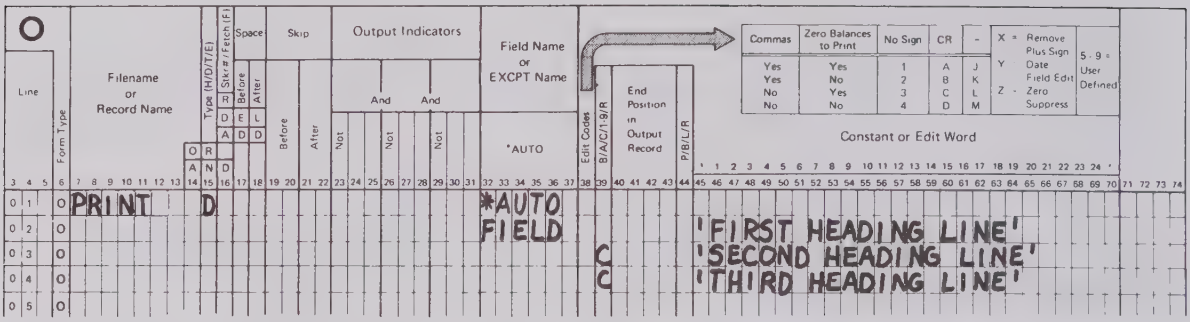
Enter a `C` in position 39. One or two C-type specifications can follow a field description specification that has an A, B, or blank in position 39 and an entry in positions 32 through 37. The first C-type specification causes a second column heading line to be generated. The second C-type specification causes a third column heading line to be generated (see Figure 8-7 on page 8-20).

Positions 40-44

Positions 40 through 44 must be blank on a C-type field description specification.

Positions 45-70 (Constant)

Enter a constant, up to 24 positions long including blanks, enclosed in apostrophes. The constant becomes the second or third line of column headings, depending on whether it is on the first or second C-type specification. If two or three column heading lines are specified, the shorter literals are centered on the longest.



C in position 39 is used to specify second and third column heading lines. A maximum of three column heading lines (two C-type field descriptions) can be used.

Figure 8-7. Specifying Second and Third Column Heading Lines

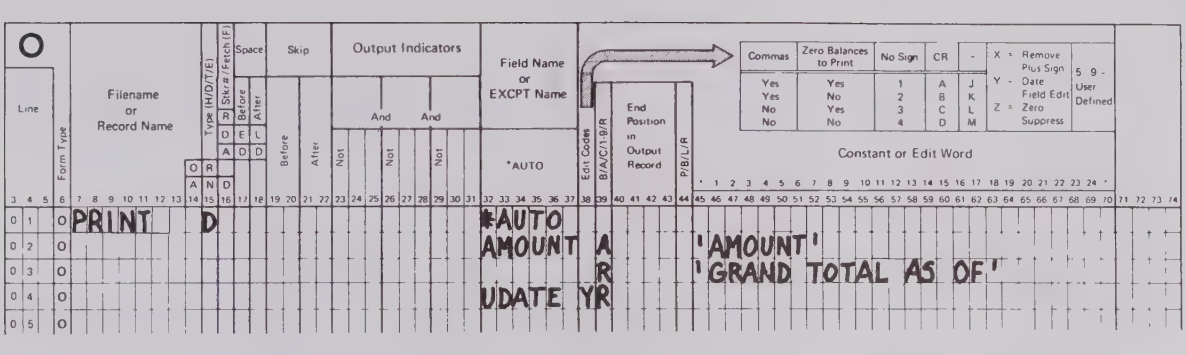
Field Description (1-9 or R in Position 39)

Enter a digit (1 through 9) or R in position 39 of a field description to specify a field or constant to be printed on a specific total line.

Auto report allows you to print other information on generated total lines in addition to the generated totals resulting from A-type field descriptions. The value entered in position 39 corresponds to the level of the total line on which the information is to be printed (the corresponding control level must be defined in positions 59 and 60 of the input specifications). For example, a 3 in position 39 indicates that the information is printed on the L3 total line; an R indicates that the information appears on the final total, or LR, line (see Figure 8-8 on page 8-21). Fields and constants specified in this way are printed to the left of the leftmost generated total on the line.

This type of field description can print information such as DISTRICT TOTAL, GRAND TOTAL, or other literal information. It can also print a field and specify an edit word, floating dollar sign, or asterisk protection for the field.

If none of the *AUTO output fields are defined with an A in position 39, then 1 through 9 or R cannot be used in position 39. In group printing, only specify numbers that are higher than the lowest control level indicator used to condition the T-*AUTO specification. If the T-*AUTO specification is not conditioned by a control level indicator, use only numbers that are higher than the lowest control level defined in positions 59 and 60 of the input specifications.



In this example, the literal 'GRAND TOTAL AS OF' followed by the current date prints on the left of the generated final total line, as shown below.

```
                                          AMOUNT

                                         XXX.XX
                                         XXX.XX
                                       X,XXX.XX

GRAND TOTAL AS OF  1/31/80          74,341.50*
```

Figure 8-8. Specifying a Constant and a Field to Print on a Generated Total Line

Positions 7-31

Positions 7 through 31 must be blank on a field description line with 1 through 9 or R in position 39.

Positions 32-37 (Field Name)

Enter the name of a field, an indexed array name, or a table name in positions 32 through 37. The corresponding field or element value prints on the total line indicated by the entry in position 39. If positions 32 through 37 are blank, a constant must be entered in positions 45 through 70.

Position 38 (Edit Code)

Enter an edit code in position 38 to edit a numeric field named in positions 32 through 37, or leave position 38 blank. If position 38 is left blank, an edit word can be entered in positions 45 through 70. If position 38 is blank, no edit code is assumed by auto report.

Position 39

Enter a digit (1 through 9) or R. These entries correspond to the indicators L1, L2, . . . L9, and LR. The entry identifies a specific total line on which the field or constant described is to be printed. The entry in position 39 must correspond to a control level that is defined by the input specifications. In group printing, the entry in this position must be higher than the control level of the first total line generated.

Positions 40-43 (End Position in Output Record)

Do not make an entry in positions 40 through 43 on field description specifications with 1 through 9 or R in position 39.

Position 44

Leave position 44 blank.

Positions 45-70 (Constant or Edit Word)

Leave positions 45 through 70 blank, or enter a constant or edit word. If field name (positions 32 through 37) on this specification line contains an entry, then positions 45 through 70 can contain any of the following:

- Blanks, if no editing is needed for the field or if the field is already edited by an edit code in position 38
- Edit word, if special editing is desired
- Floating currency symbol or asterisk protection entry used with an edit code.

Positions 45 through 70 cannot contain a constant when field name contains an entry. However, if field name is blank, positions 45 through 70 must contain a constant.

/COPY Statement

The format of the /COPY statement is:

Position	Entry
1-5	Page and line number indicating the placement of the statement in the sequence of auto report source specifications.
6	This position can contain any entry except H or U, or can be blank.
7-11	Enter the characters /COPY.
12	Blank.
13-44	Enter the qualified file name (library-name/file-name in the AS/400 Native Environment, or file-name.library-name in the System/38 Environment), followed by a comma, followed by the member name. If the library name is not specified, the library list (*LIBL) is used to locate the file. If F1, F2, R1, or R2 is specified as the file name, the file name QRPGSRC is assumed and the library list is used to locate the file. If only one entry appears, it is the member name; the file name QRPGSRC is assumed and the library list is used to locate the file. The member name must exist in QRPGSRC.
45-49	Blank.
50-80	Enter any information or comments. The contents of these positions are not read by auto report.

See *RPG/400 User's Guide* for more information on /COPY.

Chapter 9. Operation Codes

The RPG/400 language allows you to do many different types of operations on your data. Operation codes, which are entered on the calculation specifications, indicate the operation to be done. Usually these codes are abbreviations of the name of the operation.

Many operation codes can be placed into categories. The first part of this chapter includes general information about these categories. The latter part of the chapter describes each operation code in alphabetical order and shows one or more examples for most of the operations.

The tables on the next few pages are a summary of the specifications for each operation code.

Table 9-1. Summary of Operation Code Specifications 1

Operation Code[1]	Conditioning Indicators[2] Positions 9-17	Factor 1	Factor 2	Result Field	Resulting Indicators Positions 54-55	Resulting Indicators Positions 56-57	Resulting Indicators Positions 5-59
ACQ	O	R	R			O: ER	
ADD[3]	O	O	R	R	O: +	O: -	O: Z
ANDxx		R	R				
BEGSR		R					
BITOF	O		R	R			
BITON	O		R	R			
CABxx	O	R	R	O	O: HI	O: LO	O: EQ
CALL	O		R	O		O: ER	O
CASxx	O	O	O	R	O: HI	O: LO	O: EQ
CHAIN	O	R	R	O	R: NR	O: ER	
CLOSE	O		R			O: ER	
COMIT	O	O				O: ER	
COMP	O	R	R		O: HI[4]	O: LO[4]	O: EQ[4]
DEBUG	O	O	O	O			
DEFN		R	O	R			
DELET	O	O	R		O: NR	O: ER	
DIV[3]	O	O	R	R	O: +	O: -	O: Z
DO	O	O	O	O			
DOUxx	O	R	R				
DOWxx	O	R	R				
DSPLY	O	O	O	O		O: ER	
DUMP	O	O					
ELSE							
END	O[5]		O				
ENDSR		O	O				
EXCPT	O		O				
EXFMT	O		R			O: ER	
EXSR	O		R				
FEOD	O		R			O: ER	
FORCE	O		R				
FREE	O		R			O: ER	

[1]Control level indicators (L1 through L9) in positions 7 and 8 are O (optional) for every operation code, **but not in subroutines.** On all subroutine lines, from BEGSR through ENDSR, the only valid entries in positions 7 and 8 are SR, AN, OR, or blanks.
[2]Conditioning indicators are valid only for executable operation codes.
[3]Half adjust (position 53) can be specified for this operation.
[4]At least one resulting indicator must be specified in positions 54 through 59.
[5]The END operation code can have a conditioning indicator if it is part of a DO, DOUxx, OR DOWxx group, but not if it is part of a CASxx or IFxx group. Fields without entries must be blank.

+ = Plus - = Minus BOF = Beginning of file EOF = End of file	EQ = Equal ER = Error NR = No record found O = Optional	R = Required Z = Zero ZB = Zero or blank

Table 9-2. Summary of Operation Code Specifications 2

Operation Code[1]	Conditioning Indicators[2] Positions 9-17	Factor 1	Factor 2	Result Field	Resulting Indicators Positions 54-55	Resulting Indicators Positions 56-57	Resulting Indicators Positions 58-59
GOTO	O		R				
IFxx	O	R	R				
IN	O	O	R			O: ER	
KFLD				R			
KLIST		R					
LOKUP (Array)	O	R	R		O: HI[4]	O: LO[4]	O: EQ[4]
LOKUP (Table)	O	R	R	O	O: HI[4]	O: LO[4]	O: EQ[4]
MHHZO	O		R	R			
MHLZO	O		R	R			
MLHZO	O		R	R			
MLLZO	O		R	R			
MOVE	O		R	R	O: +	O: -	O: ZB
MOVEA	O		R	R	O: +	O: -	O: ZB
MOVEL	O		R	R	O: +	O: -	O: ZB
MULT[3]	O	O	R	R	O: +	O: -	O: Z
MVR	O			R	O: +	O: -	O: Z
NEXT	O	R	R			O: ER	
OCUR	O	O	R	O		O: ER	
OPEN	O		R			O: ER	
ORxx		R	R				
OUT	O	O	R			O: ER	
PARM		O	O	R			
PLIST		R					
POST	O	O	O[6]	O[6]		O: ER	

[1]Control level indicators (L1 through L9) in positions 7 and 8 are O (optional) for every operation code, **but not in subroutines**. On all subroutine lines, from BEGSR through ENDSR, the only valid entries in positions 7 and 8 are SR, AN, OR, or blanks.
[2]Conditioning indicators are valid only for executable operation codes.
[3]Half adjust (position 53) can be specified for this operation.
[4]At least one resulting indicator must be specified in positions 54 through 59.
[5]The END operation code can have a conditioning indicator if it is part of a DO, DOUxx, OR DOWxx group, but not if it is part of a CASxx or IFxx group.
[6]You must specify factor 2 or the result field. You may specify both.

Fields without entries must be blank.

+ = Plus - = Minus BOF = Beginning of file EOF = End of file	EQ = Equal ER = Error NR = No record found O = Optional	R = Required Z = Zero ZB = Zero or blank

Table 9-3. Summary of Operation Code Specifications 3

Operation Code[1]	Conditioning Indicators[2] Positions 9-17	Factor 1	Factor 2	Result Field	Resulting Indicators Positions 54-55	Resulting Indicators Positions 56-57	Rsulting Indicators Positions 58-59
READ	O		R	O		O: ER	R: EOF
READC	O		R			O: ER	R: EOF
READE	O	R	R	O		O: ER	R: EOF
READP	O		R	O		O: ER	R: BOF
REL	O	R	R			O: ER	
RETRN	O						
ROLBK	O					O: ER	
SETGT	O	R	R		O: NR	O: ER	
SETLL	O	R	R		O: NR	O: ER	O: EQ
SETOF	O				O[4]	O[4]	O[4]
SETON	O				O[4]	O[4]	O[4]
SHTDN	O				R		
SORTA	O		R				
SQRT[3]	O		R	R			
SUB[3]	O	O	R	R	O: +	O: -	O: Z
TAG		R					
TESTB	O		R	R	O[4]	O[4]	O[4]
TESTN	O			R	O[4]	O[4]	O[4]
TESTZ	O			R	O[4]	O[4]	O[4]
TIME	O			R			
UNLCK	O		R			O: ER	
UPDAT	O		R	O		O: ER	
WRITE	O		R	O		O: ER	O: EOF
XFOOT[3]	O		R	R	O: +	O: -	O: Z
Z-ADD[3]	O		R	R	O: +	O: -	O: Z
Z-SUB[3]	O		R	R	O: +	O: -	O: Z

[1]Control level indicators (L1 through L9) in positions 7 and 8 are O (optional) for every operation code, **but not in subroutines.** On all subroutine lines, from BEGSR through ENDSR, the only valid entries in positions 7 and 8 are SR, AN, OR, or blanks.
[2]Conditioning indicators are valid only for executable operation codes.
[3]Half adjust (position 53) can be specified for this operation.
[4]At least one resulting indicator must be specified in positions 54 through 59.
[5]The END operation code can have a conditioning indicator if it is part of a DO, DOUxx, OR DOWxx group, but not if it is part of a CASxx or IFxx group.

Fields without entries must be blank.

+ = Plus - = Minus BOF = Beginning of file EOF = End of file	EQ = Equal ER = Error NR = No record found O = Optional	R = Required Z = Zero ZB = Zero or blank

Arithmetic Operations

The arithmetic operations are:

- ADD (Add)
- DIV (Divide)
- MULT (Multiply)
- MVR (Move Remainder)
- SQRT (Square Root)
- SUB (Subtract)
- XFOOT (Summing the Elements of an Array)
- Z-ADD (Zero and Add)
- Z-SUB (Zero and Subtract).

Remember the following when specifying arithmetic operations:

- Arithmetic operations can be done only on numeric fields (including numeric subfields and numeric PARM fields, numeric arrays, numeric array elements, numeric table elements, and numeric literals).
- Arithmetic operations are done on data in packed decimal format. Data maintained in other formats is converted to or from packed decimal format.
- Decimal alignment is done for all arithmetic operations. Even though truncation can occur, the position of the decimal point in the result field is not affected.
- Factor 1 and factor 2 are not changed by an arithmetic operation.
- Any data placed in the result field replaces the data that was there previously.
- The result field must be large enough to accommodate the results of the arithmetic operation because an RPG/400 program does not cause an error on an arithmetic overflow. If the result field is not large enough to accommodate the results, digits are dropped from either or both ends, depending on the location of the decimal point.
- If you use conditioning indicators, it is your responsibility to ensure that the DIV operation is done immediately before the MVR operation. If the MVR operation is done before the DIV operation, undesirable results occur. For example, error message RPG0907 may be issued.
- Half adjust (position 53) can be specified for all the arithmetic operations except for the MVR operation or for a DIV operation that is immediately followed by an MVR operation.

For arithmetic operations in which all three fields are used:

- Factor 1, factor 2, and the result field can be three different fields.
- Factor 1, factor 2, and the result field can all be the same field.
- Factor 1 and factor 2 can be the same field but different from the result field.
- Either factor 1 or factor 2 can be the same as the result field.

The length of any field specified in an arithmetic operation cannot exceed 30 digits. If the result exceeds 30 digits, digits are dropped from either or both ends depending on the location of the decimal point.

All arithmetic operations are done algebraically.

The results of all operations are signed (a plus sign is a hexadecimal F and a minus sign is a hexadecimal D) according to the following rules. The sign is in the zone portion of the low-order byte.

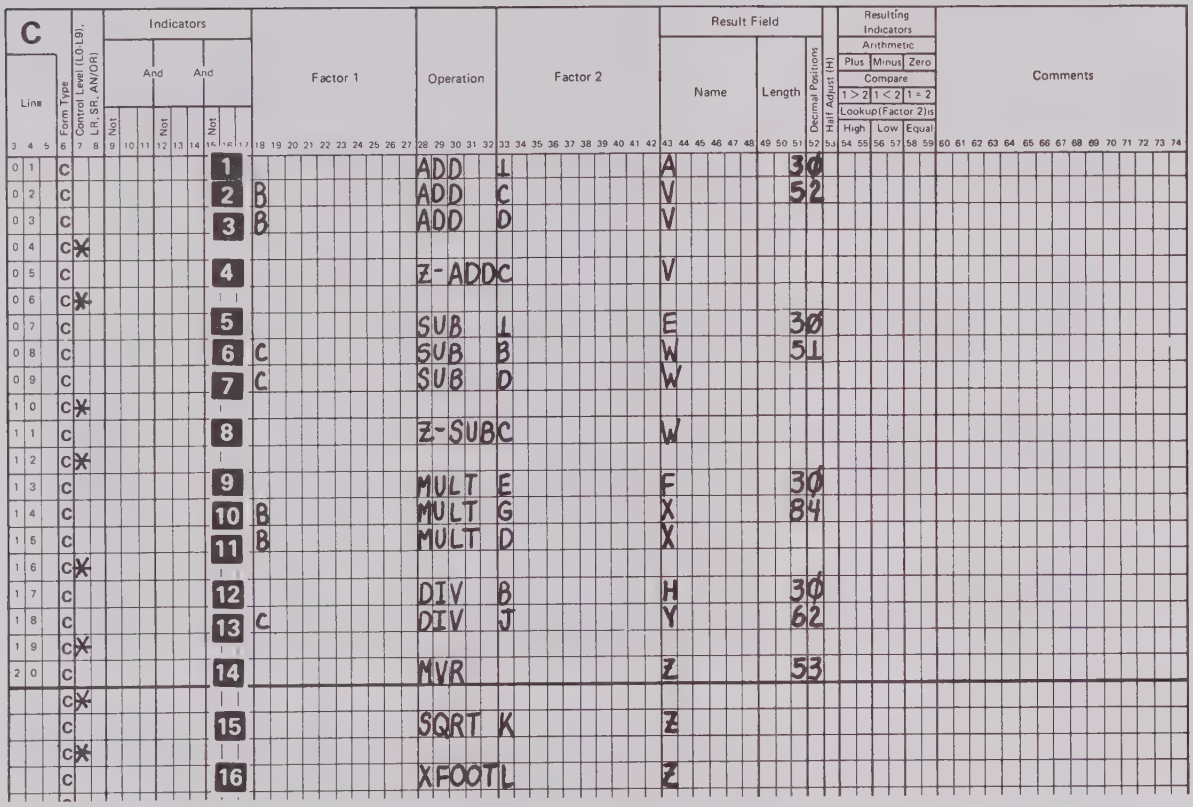
Addition: If factor 1 and factor 2 have like signs, the result field sign is the same. If factor 1 and factor 2 have unlike signs, the result field sign is the same as the sign of the larger factor.

Subtraction: Change the sign of factor 2 and apply the rules for addition.

Multiplication: If factor 1 and factor 2 have like signs, the result field sign is plus (+). If factor 1 and factor 2 have unlike signs, the result field sign is minus (-).

Division: If factor 1 and factor 2 have like signs, the result field sign is plus (+). If factor 1 and factor 2 have unlike signs, the result field sign is minus (-). The sign of the remainder is the same as the factor 1 sign.

For the ADD, SUB, MULT, and DIV operations, factor 1 is not required. If factor 1 is not specified, the operation is done as though factor 1 and the result field were the same field.



	Factor 1	Operation	Factor 2	Result Field Name	Result Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic Plus	Arithmetic Minus	Arithmetic Zero	Compare 1 > 2	Compare 1 < 2	Compare 1 = 2	Lookup (Factor 2) is High	Lookup (Factor 2) is Low	Lookup (Factor 2) is Equal	Comments
1		ADD L		A	30												
2	B	ADD C		V	52												
3	B	ADD D		V													
4		Z-ADD C		V													
5		SUB L		E	30												
6	C	SUB B		W	51												
7	C	SUB D		W													
8		Z-SUB C		W													
9		MULT E		F	30												
10	B	MULT G		X	84												
11	B	MULT D		X													
12		DIV B		H	30												
13	C	DIV J		Y	62												
14		MVR		Z	53												
15		SQRT K		Z													
16		XFOOT L		Z													

Before the operations are processed, the field values are:

```
A = 1.00          G = 2.77
B = 10.0          H = 70
C = 32            J = .6
D = -20           K = 25
E = 6.            L = 1.0,1.7,-1.1
F = 10.0
```

After the operations are processed, the result fields contain the following values:

```
1  002            9  060
2  042.00        10  0027.7000
3  -10.00        11  -200.0000
4  032.00        12  007
5  005           13  0053.33
6  0022.0        14  00.002
7  0052.0        15  0.5000
8  -0032.0       16  01.600
```

Figure 9-1. Summary of Arithmetic Operations

Move Operations

The move operations are:

- MOVE (Move)
- MOVEA (Move Array)
- MOVEL (Move Left).

Move operations transfer all or part of factor 2 to the result field. Factor 2 remains unchanged. Factor 1 must be blank. Resulting indicators can be specified in positions 54 through 59 except for the MOVE and MOVEL operations if the result field is an array, or for the MOVEA operation if the result field is an array or array element.

In the move operations, factor 2 and the result field are generally of the same type (both numeric or both character). However, the move operations can also be used to change numeric fields to character fields and character fields to numeric fields. To change a numeric field to a character field, enter the name of the numeric field in factor 2 and specify a character result field. To change a character field to a numeric field, enter the name of the character field in factor 2 and specify a numeric result field.

When a character field is moved into a numeric result field, the digit portion of each character is converted to its corresponding numeric character and then moved to the result field. Blanks are transferred as zeros. For the MOVE operation, the zone portion of the rightmost character is converted to its corresponding sign and is moved to the rightmost position of the numeric result field where it becomes the sign of the field (see Figure 9-25 on page 9-90 for an example of this). For the MOVEL operation, the zone portion of the rightmost character of factor 2 is converted and used as the sign of the result field (unless factor 2 is shorter than the result field) whether or not the rightmost character is included in the move operation (see Figure 9-27 on page 9-97 for an example of this).

If move operations are specified to move data into numeric fields, the decimal positions specified for the factor 2 field are ignored. For example, if 1.00 is moved into a three-position numeric field with one decimal position, the result is 10.0.

When resulting indicators are specified for move operations, the result field determines which indicator is set on. If the result field is a character field, only the resulting indicator in positions 58 and 59 can be specified. This indicator is set on if the result field is all blanks. When the result field is numeric, all three resulting indicator positions may be used. These indicators are set on as follows:

High	Set on if the result field is greater than 0.
Low	Set on if the result field is less than 0.
Equal	Set on if the result field is equal to 0.

Move Zone Operations

The move zone operations are:

- MHHZO (Move High to High Zone)
- MHLZO (Move High to Low Zone)
- MLHZO (Move Low to High Zone)
- MLLZO (Move Low to Low Zone).

The move zone operations move only the zone portion of a character.

A minus (-) sign in a move zone operation does not yield a negative character in the result field because a minus sign is represented by a hexadecimal 60 internally and a D zone is required for a negative character. Characters J through R have D zones and can be used to obtain a negative value

(J = hexadecimal D1, ..., R = hexadecimal D9).

Note: Whenever the word *high* is used in a move zone operation, the field involved must be a character field; whenever *low* is used, the field involved can be either a character or a numeric field.

Compare Operations

The compare operations are:

- COMP (Compare)
- CABxx (Compare and Branch)
- CASxx (Case).

The compare operations test fields for certain conditions. Resulting indicators assigned in positions 54 through 59 are set according to the results of the operation, or a branch occurs based on the results of the operation. No fields are changed by these operations.

Remember the following when using the compare operations:

- If numeric fields are compared, fields of unequal length are aligned at the implied decimal point. The shorter field is filled with zeros to the left or right of the decimal point so that the field lengths are equal for comparison.
- If character fields are compared, fields of unequal length are aligned to their left-most character. The shorter field is filled with blanks to equal the length of the longer field so that the field lengths are equal for comparison.
- All numeric comparisons are algebraic. A plus (+) value is always greater than a minus (-) value.
- Blanks within numeric fields are assumed to be zeros.
- Numeric fields are converted to packed decimal format, if necessary, before they are compared.
- If an alternate collating sequence (position 26 of the control specification) has been specified for the comparison of character fields, the fields are translated to

the alternate sequence and then compared. If *HIVAL and *LOVAL are used to set up the compare, the alternate collating sequence may alter the value prior to the compare operation.

- A character field cannot be compared with a numeric field.
- An array name cannot be specified in a compare operation.

Declarative Operations

The declarative operations do not run, these operations that can be specified anywhere within calculations. The control level entry (positions 7 and 8) can be blank or can contain an entry to group the statements within the appropriate section of the program. The control level entry is for documentation purposes only. The declarative operations are:

- DEFN (Field Definition)
- KFLD (Define Parts of a Key)
- KLIST (Define a Composite Key)
- PARM (Identify Parameters)
- PLIST (Identify a Parameter List)
- TAG (Tag).

The DEFN operation either defines a field based on the attributes (length and decimal positions) of another field or defines a field as a data area.

The KLIST and KFLD operations are used respectively to indicate the name by which a composite key field may be referenced and the fields that compose the composite key. A *composite key* is a key that contains more than one key field. A composite key is built from left to right, with the first KFLD specified being the leftmost (high-order) field of the composite key.

The PLIST and PARM operations are used with the CALL operation to allow a called program to access parameters in a calling program.

The TAG operation names the destination of a branching operation such as GOTO or CABxx.

Subroutine Operations

An RPG/400 subroutine is a group of calculation specification statements in a program that can be processed several times in that program. The RPG/400 subroutine operations are:

- BEGSR (Begin Subroutine)
- ENDSR (End Subroutine)
- EXSR (Run Subroutine).

RPG/400 subroutine specifications must follow all other calculation operations that can be processed for a program; however, the PLIST, PARM, KLIST, KFLD, and DEFN operations may be specified between an ENDSR operation (the end of one subroutine)

and a BEGSR operation (the beginning of another subroutine) or after all subroutines. A subroutine can be done from any point in the calculation operations. Subroutine lines can be identified by SR in positions 7 and 8. The only valid entries in positions 7 and 8 of a subroutine line are SR., AN, OR, or blanks.

For information on how to code a subroutine see "Coding Subroutines" on page 9-67.

SETON and SETOF Operations

The SETON and SETOF operations set (on or off) indicators specified in positions 54 through 59. At least one resulting indicator must be specified in these positions. Remember the following when setting indicators:

- The 1P, MR, KA through KN, and KP through KY indicators cannot be set on by the SETON operation.
- The 1P and MR indicators cannot be set off by the SETOF operation.
- Setting L1 through L9 on or off with a SETON or SETOF operation does not automatically set any lower control level indicators.

Structured Programming Operations

The structured programming operations are:

- ANDxx (And)
- CASxx (Case)
- DO (Do)
- DOUxx (Do Until)
- DOWxx (Do While)
- ELSE (Else Do)
- END (End)
- IFxx (If/Then)
- ORxx (Or).

The DO operation allows the processing of a group of calculations one or more times starting with the value in factor 1, incrementing each time by a value on the associated END operation until the limit specified in factor 2 is reached.

The DOUxx and DOWxx operations allow the processing of a group of calculations one or more times based on the results of comparing factor 1 and factor 2.

An IFxx operation allows the processing of a group of calculations one or more times based on the results of comparing factor 1 and 2.

The ANDxx and ORxx operations are used with the DOUxx, DOWxx, and IFxx operations to specify a more complex condition than the comparison of a single factor 1 and factor 2.

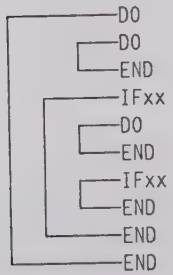
A DO, DOUxx, DOWxx, or IFxx operation (with or without ANDxx or ORxx operations), and an END operation, delimit a do group. The END operation, called the associated END of the do group, ends each do group or causes the do group to be reprocessed until the specified ending conditions are met. An IFxx operation and an IFxx operation with an ELSE operation must also end with an END operation.

The rules for how the comparison is made on the ANDxx, DOUxx, DOWxx, IFxx, and ORxx operation codes are the same as those given under "Compare Operations" on page 9-9.

In the ANDxx, CASxx, DOUxx, DOWxx, IFxx, and ORxx operations, xx can be:

xx	Meaning
GT	Factor 1 is greater than factor 2.
LT	Factor 1 is less than factor 2.
EQ	Factor 1 is equal to factor 2.
NE	Factor 1 is not equal to factor 2.
GE	Factor 1 is greater than or equal to factor 2.
LE	Factor 1 is less than or equal to factor 2.
Blanks	Unconditional processing (CASxx only).

If a do group contains another complete do group, together they form a nested do group. Do groups can be nested to a maximum depth of 100 levels. The following is an example of nested do groups, three levels deep:



Remember the following when specifying do groups:

- Each nested do group must be contained within the outer level do group.
- Each do group must contain both a DO, DOUxx, DOWxx, or IFxx operation and an END operation.
- A do group can be contained in either detail or total calculations; it cannot be split between the two.
- Branching into a do group from outside the do group may yield undesirable results.

Call and Branching Operations

The call and branching operations are:

- CALL (Call a Program)
- EXCPT (Calculation Time Output)
- FREE (Deactivate a Program)
- GOTO (Branch To)
- RETRN (Return to Caller).

The CALL/FREE/RETRN operations allow an RPG/400 program to transfer control to other programs. With the CALL/FREE/RETRN function, the PLIST and PARM operations that do not run can be used to allow the calling and called programs to address the same data.

The GOTO operation (when used with a TAG operation) allows the programmer to done calculation specifications in an order different from the order specified on the calculation specifications. For example:

- Several operations can be skipped when certain conditions occur.
- Certain operations can be done for several, but not all, record types.
- Several operations can be repeated.

The EXCPT operation allows records to be written during calculation time instead of output time.

File Operations

The file operation codes are:

- ACQ (Acquire)
- CHAIN (Random Retrieval from a File Based on Record Number or Key Value)
- CLOSE (Close File)
- COMIT (Commit)
- DELET (Delete Record)
- EXFMT (Execute Format)
- FEOD (Force End of Data)
- NEXT (Next)
- OPEN (Open File for Processing)
- POST (Post)
- READ (Read a Record)
- READC (Read Next Modified Record)
- READE (Read Equal Key)
- READP (Read Prior Record)
- REL (Release)
- ROLBK (Roll Back)
- SETGT (Set Greater Than)
- SETLL (Set Greater Than or Equal To)
- UPDAT (Modify Existing Record)
- WRITE (Create a New Record).

File operations can be used with both program described and externally described files (F or E respectively in position 19 of the file description specifications).

When an externally described file is used with certain file operations, a record format name, rather than a file name, can be specified in factor 2. Thus, the processing operation code retrieves and/or positions the file at the first record format of the specified type according to the rules of the calculation operation code used.

The WRITE and UPDAT operations that specify a program described file name in factor 2 *must* have a data structure name specified in the result field. The CHAIN, READ, READE, and READP operations that specify a program described file name in factor 2 *may* have a data structure name specified in the result field. With the CHAIN, READ, READE, and READP operations, data is transferred directly between the file and the data structure, bypassing the normal field extract function. Thus, no record identifying or field indicators are set on as a result of an input operation to a data structure. If all input and output operations to the file have a data structure specified in the result field, input and output specifications are not required. However, the data structure must be defined on the input specifications. If an input operation (CHAIN, EXFMT, READ, READC,.READE, READP) does not retrieve a record because no record was found, because an error occurred in the operation, or because the last record was already retrieved (end of file), then RPG/400 does not extract any data and all fields in the program remain unchanged.

Exception/errors that occur during file operations can be handled by the programmer or by the RPG/400 default error handler.

Information Operations

The information operations are:

- DEBUG (Debug Function)
- DUMP (Program Dump)
- SHTDN (Shut Down)
- TIME (Time of Day).

The DEBUG operation writes the following information that you can use to debug an RPG/400 program:

- The source statement sequence number of the DEBUG operation
- All indicators that are on at the time the operation is encountered
- The contents of factor 1 (if factor 1 is specified) that identifies the DEBUG written information
- The contents of the result field (if the result field contains an entry).

The information is sent to the file named in factor 2, or, if factor 2 is blank, the information is sent to the requester.

The DUMP operation provides a dump of all indicators, fields, data structures, arrays, and tables used in a program.

The SHTDN operation allows the program to determine whether the system operator has requested shutdown. If so, the resulting indicator that must be specified in positions 54 and 55 is set on.

The TIME operation allows the program to access the system time of day and system date at any time during program running.

Message Operation

The message operation, DSPLY, allows communication between the program and the system console or between the program and the display work station that requested the program.

Data Area Operations

The data area operations are:

- IN (Retrieve a Data Area)
- OUT (Write a Data Area)
- UNLCK (Unlock a Data Area).

The IN and OUT operations allow you to retrieve and write one or all data areas in a program, depending on the factor 2 entry.

The IN and OUT operations also allow you to control the locking or unlocking of a data area. When a data area is locked, it cannot be accessed by other programs; when it is unlocked, it can be accessed by other programs.

RPG/400 uses the following lock states:

- For an IN operation with *LOCK specified, RPG/400 places an *EXCLRD (exclusive allow read) lock state on the data area.
- For an OUT or UNLCK operation, RPG/400 releases the *EXCLRD lock state.

During the actual transfer of data into or out of a data area, there is also a system-internal lock on the data area. If several users are contending for the same data area, it is possible for a user to get the error message RPG0431 indicating that the data area is not available.

Remember the following when using the IN, OUT, and UNLCK operations:

- A data area operation cannot be done on a data area that is not defined to OS/400.
- Before the IN, OUT, and UNLCK operations can be done on a data area, you must specify that data area in the result field of an *NAMVAR DEFN statement. (For further information on the DEFN statement, see "DEFN (Field Definition)" on page 9-42.)

- The data area operations can be done on a data area data structure that is implicitly retrieved by RPG/400 only if the data area data structure name is specified in the result field of an *NAMVAR DEFN statement.
- A locked data area cannot be updated or locked by another RPG/400 program; however, the data area can be retrieved in your own program by an IN operation with factor 1 blank.
- A data area name cannot be the name of a multiple occurrence data structure, an input record field, an array, an array element, or a table.
- A data area cannot be the subfield of a multiple occurrence data structure, a data area data structure, a program status data structure, a file information data structure (INFDS), or a data structure that appears on an *NAMVAR DEFN statement.

OPERATION CODES

The remainder of this chapter describes, in alphabetical order, each operation code.

ACQ (Acquire)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Blank	Blank	Opt: ER	Blank

The ACQ operation acquires the program device specified in factor 1 for the WORKSTN file specified in factor 2. If the device is available, ACQ attaches it to the file. If it is not available or is already attached to the file, an error occurs. If an indicator is specified in positions 56 and 57, the indicator is set on. If no indicator is specified, but the INFSR subroutine is specified, the INFSR automatically receives control when an error/exception occurs. If no indicator or INFSR subroutine is specified, the default error/exception handler receives control when an error/exception occurs.

No input or output operation occurs when the ACQ operation is processed. ACQ must only be used with a multiple device file.

ADD (Add)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Required	Required	Opt: +	Opt: –	Opt: Z

If factor 1 is specified, factor 1 and factor 2 are added together and the result is placed in the result field. If factor 1 is not specified, the contents of factor 2 are added to the contents of the result field. For the rules for specifying an ADD operation, see "Arithmetic Operations" on page 9-5.

See Figure 9-1 on page 9-7 for examples of the ADD operation.

ANDxx (And)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Blank	Required	Required	Blank	Blank	Blank	Blank

The ANDxx operation is optional with the DOUxx, DOWxx, IFxx, and ORxx operations. ANDxx is specified immediately following a DOUxx, DOWxx, IFxx, ORxx, or ANDxx statement. The ANDxx operation allows a more complex condition to be specified for the DOUxx, DOWxx, and IFxx operations.

The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry must be the same as the control level entry for the associated DOUxx, DOWxx, or IFxx operation. Conditioning indicator entries (positions 9 through 17) are not permitted.

Factor 1 and factor 2 must contain a character literal, a numeric literal, or a field name. Factor 1 and factor 2 must be either both character data or both numeric data. The comparison of factor 1 and factor 2 follows the same rules as those given for the compare operations. See "Compare Operations" on page 9-9.

Figure 9-12 on page 9-51 shows an example of ANDxx and ORxx operations with a DOUxx operation.

BEGSR (Begin Subroutine)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt SR	Blank	Required	Blank	Blank	Blank	Blank	Blank

The BEGSR operation identifies the beginning of an RPG/400 subroutine. Factor 1 contains the name of the subroutine. The same name must be specified in factor 2 of the EXSR operation referring to the subroutine or in the result field of the CASxx operation referring to the subroutine. The control level entry (positions 7 and 8) can be SR or remain blank. Conditioning indicator entries are not permitted.

Every subroutine must have a unique symbolic name. The keyword *PSSR used in factor 1 specifies that this is a program exception/error subroutine to handle program-detected exception/errors. Only one subroutine can be defined by this keyword in factor 1.

BITOF (Set Bits Off)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Blank	Blank	Blank

The BITOF operation causes bits identified in factor 2 to be set off (be set to 0) in the field named as the result field. The bits that are not identified in factor 2 remain unchanged. Therefore, when using BITOF to format a character, you should use both BITON and BITOF. BITON to specify the bits to be set on (= 1), and BITOF to specify the bits to be set off (= 0). This will ensure that any previously set bits in the field do not remain to alter the character format you wish to specify.

Factor 2 can contain:

- *Bit numbers 0-7:* From 1 to 8 bits can be set off per operation. The bits to be set off are identified by the numbers 0 through 7 (0 is the leftmost bit). The bit numbers must be enclosed in apostrophes, and the entry must begin in position 33. For example, to set off bits 0, 2, and 5, enter '025' in factor 2.
- *Field name:* The name of a one-position character field, table element, or array element can be specified in factor 2. In this case, the bits that are on in the field, table element, or array element are set off in the result field; bits that are off are not affected.

Any field named in the result field must be a one-position character field. The field specified in the result field can be an array element if each element in the array is a one-position character field.

See Figure 9-2 on page 9-23 for an example of the BITOF operation.

BITON (Set Bits On)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Blank	Blank	Blank

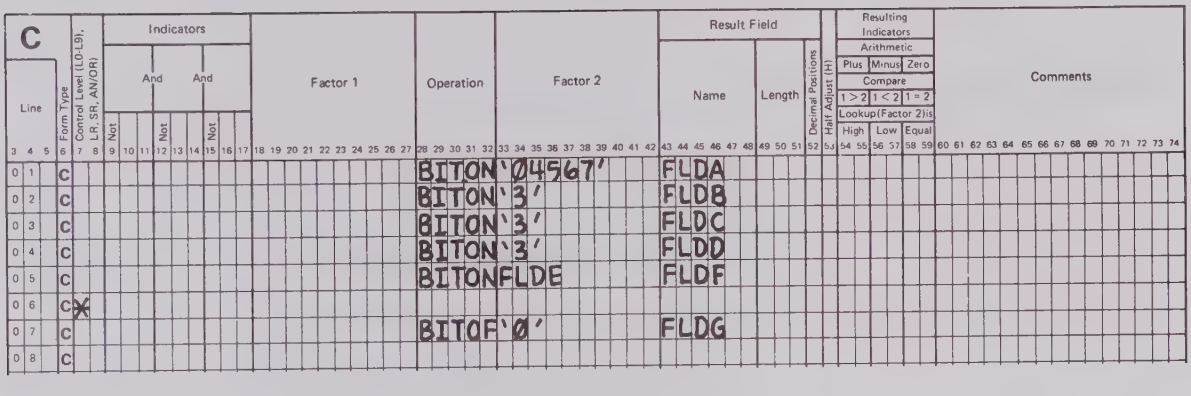
The BITON operation causes bits identified in factor 2 to be set on (be set to 1) in the field named as the result field. The bits that are not identified in factor 2 remain unchanged. Therefore, when using BITON to format a character, you should use both BITON and BITOF. BITON to specify the bits to be set on (= 1), and BITOF to specify the bits to be set off (= 0). This will ensure that any previously set bits in the field do not remain to alter the character format you wish to specify.

Factor 2 can contain:

- *Bit numbers 0-7:* From 1 to 8 bits can be set on per operation. The bits to be set on are identified by the numbers 0 through 7 (0 is the leftmost bit). The bit numbers must be enclosed in apostrophes, and the entry must begin in position 33. For example, to set bits 0, 2, and 5 on, enter '025' in factor 2.
- *Field name:* The name of a one-position character field, table element, or array element can be specified in factor 2. In this case, the bits that are on in the field, table element, or array element are set on in the result field; bits that are off are not affected.

Any field named in the result field must be a one-position character field. The field specified in the result field can be an array element if each element in the array is a one-position character field.

Figure 9-2 on page 9-23 illustrates uses of the BITON operation.



Before the operations are processed, the bit settings are:

```
FLDA = 00000000
FLDB = 00000000
FLDC = 11111111
FLDD = 11000000
FLDE = 11000000
FLDF = 10000001
FLDG = 11111111
```

As a result of the BITON and BITOF operations:

```
FLDA = 10001111
FLDB = 00010000
FLDC = 11111111
FLDD = 11010000
FLDF = 11000001
FLDG = 01111111
```

Figure 9-2. BITON and BITOF Operations

CABxx (Compare and Branch)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Optional	Opt: HI	Opt: LO	Opt: EQ

The CABxx operation compares factor 1 with factor 2. If the comparison condition is true as specified by the xx portion of the operation, the program branches to the TAG operation associated with the label specified in the result field. Otherwise, the program continues with the next processable operation in the sequence. If the result field is not specified, the resulting indicators (positions 54-59) are set accordingly, and the program continues with the next processable operation in the sequence.

Conditioning indicators can be specified. Factor 1 and factor 2 must contain a character literal, a numeric literal, or a field name. Both the factor 1 and the factor 2 entries must be character data, or both must be numeric.

The CABxx operation can specify a branch:

- To a previous or a succeeding specification line
- From a detail calculation line to another detail calculation line
- From a total calculation line to another total calculation line
- From a detail calculation line to a total calculation line
- From a subroutine to a detail calculation line or to a total calculation line.

The CABxx operation cannot specify a branch from outside a subroutine to a TAG or ENDSR operation within that subroutine.

Branching from one part of the RPG/400 logic cycle to another may result in an endless loop. You are responsible for ensuring that the logic of your program does not produce undesirable results.

The xx portion of the CABxx operation can be:

xx	Meaning
GT	Factor 1 is greater than factor 2.
LT	Factor 1 is less than factor 2.
EQ	Factor 1 is equal to factor 2.
NE	Factor 1 does not equal factor 2.
GE	Factor 1 is greater than or equal to factor 2.
LE	Factor 1 is less than or equal to factor 2.

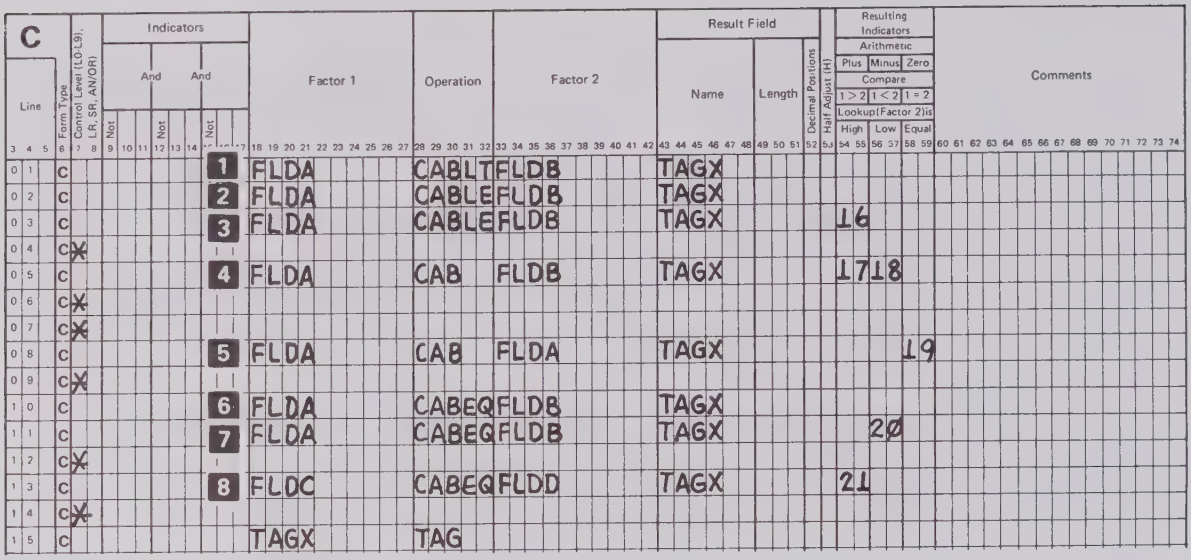
xx	Meaning
Blanks	A GOTO is done to the TAG specified in the result field, independently of the compare.

The label specified in the result field must be associated with a unique TAG operation. The label must be a unique symbolic name.

Resulting indicators are optional except when xx is blank. When xx is blank, at least one resulting indicator must be specified. When specified, the resulting indicators are set to reflect the results of the compare.

See "Compare Operations" on page 9-9 for the rules for comparing factor 1 with factor 2.

Figure 9-3 on page 9-26 illustrates uses of the CABxx operation.



Line	Form Type	Control Level (L0-L9); LR, SR, AN/OR)	Indicators: And / And		Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic: Plus	Minus	Zero; Compare: 1 > 2	1 < 2	1 = 2; Lookup (Factor 2): High	Low	Equal	Comments
01	C			1	FLDA	CABLT	FLDB	TAGX												
02	C			2	FLDA	CABLE	FLDB	TAGX												
03	C			3	FLDA	CABLE	FLDB	TAGX									16			
04	C	X																		
05	C			4	FLDA	CAB	FLDB	TAGX									17	18		
06	C	X																		
07	C	X																		
08	C			5	FLDA	CAB	FLDA	TAGX											19	
09	C	X																		
10	C			6	FLDA	CABEQ	FLDB	TAGX												
11	C			7	FLDA	CABEQ	FLDB	TAGX										20		
12	C	X																		
13	C			8	FLDC	CABEQ	FLDD	TAGX									21			
14	C	X																		
15	C				TAGX	TAG														

The field values are:

```
FLDA = 100.00
FLDB = 105.00
FLDC = ABC
FLDD = ABCDE
```

As a result of the CABxx operations the following results occur:

1. Branch to TAGX.
2. Branch to TAGX.
3. Branch to TAGX; indicator 16 is off.
4. Branch to TAGX; indicator 17 is off and indicator 18 is on.
5. Branch to TAGX; indicator 19 is on.
6. No branch occurs.
7. No branch occurs; indicator 20 is on.
8. No branch occurs; indicator 21 is off.

Figure 9-3. CABxx Operations

CALL (Call a Program)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Optional	Blank	Opt: ER	Opt

The CALL operation passes control to the program specified in factor 2.

Factor 2 must contain the name of a field, a literal, or an array element that contains the name of the program to be called and that optionally contains the name of the library in which the program is located. Factor 2 must contain a character entry. If you specify the library name, the library name must be immediately followed by a slash and then the program name (for example, 'LIB/PROG'.). The total length of a literal, including the slash, cannot exceed 8 characters. The total length of a field, including the slash, cannot exceed 21 characters. RPG/400 uses the program name exactly as specified in the literal, field, or array element to determine the program to be called. (Lowercase characters are not shifted to uppercase, and a name enclosed in quotation marks (for example "ABC") always includes the the quotation marks as part of the name of the program to be called.)

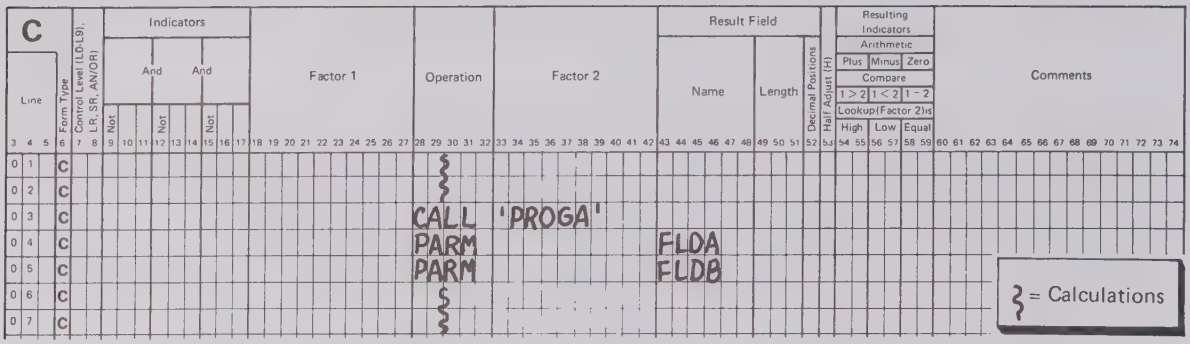
For System/38 Environment programs, if you specify the library name for the called program, the program name must be immediately followed by a period and then the library name.

In the result field, specify the name of a PLIST in order to communicate values between the calling program and the called program. The result field can be blank if the called program does not access parameters or if the PARM statements directly follow the CALL operation.

Positions 54 and 55 must be blank. Any valid resulting indicator can be specified in positions 56 and 57 to be set on for an error return from the called program. Any valid resulting indicator can be specified in positions 58 and 59 to be set on if the called program is an RPG/400 program that returns with the LR indicator on.

When a non-RPG/400 program is called, the indicators in positions 56 and 57 are set on when the called program ends in error.

Figure 9-4 on page 9-28 illustrates the use of the CALL operation.



The CALL operation calls PROGA and allows PROGA to access the parameter list containing FLDA and FLDB (lines 04 and 05). FLDA and FLDB are defined elsewhere in the program. PROGA is processed using the contents of FLDA and FLDB. After PROGA is processed, control returns to the next processable statement after the last PARM statement (line 06).

Figure 9-4. CALL Operation

CASxx (Case)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Optional	Required	Opt: HI	Opt: LO	Opt: EQ

The CASxx operation allows a subroutine to be conditionally selected for processing. The selection is based on the relationship between factor 1 and factor 2, as specified by the xx portion of the CASxx operation. If the relationship denoted by xx exists between factor 1 and factor 2, the subroutine specified in the result field is run.

Conditioning indicators can be specified. Factor 1 and factor 2 can contain a character literal, a numeric literal, a field name, or blanks (blanks are valid only for CASbb). If factor 1 and factor 2 are not blanks, both must be character data or both must be numeric. In a CASbb operation, factor 1 and factor 2 are required only if resulting indicators are specified in positions 54 through 59.

The xx portion of the CASxx operation can be:

xx	Meaning
GT	Factor 1 is greater than factor 2.
LT	Factor 1 is less than factor 2.
EQ	Factor 1 is equal to factor 2.
NE	Factor 1 is not equal to factor 2.
GE	Factor 1 is greater than or equal to factor 2.
LE	Factor 1 is less than or equal to factor 2.
Blanks	Factor 1 is not compared to factor 2 (unconditional processing).

The result field must contain the name of a valid RPG/400 subroutine. If the relationship denoted by xx exists between factor 1 and factor 2, the subroutine specified in the result field is processed. If the relationship denoted by xx does not exist, the program continues with the next CASxx operation in the CAS group. A CAS group can contain only CASxx operations. An END operation must follow the last CASxx operation to denote the end of the CAS group. After the subroutine is processed, the program continues at the next processable operation following the END operation for the CAS group, unless the subroutine passes control to a different operation.

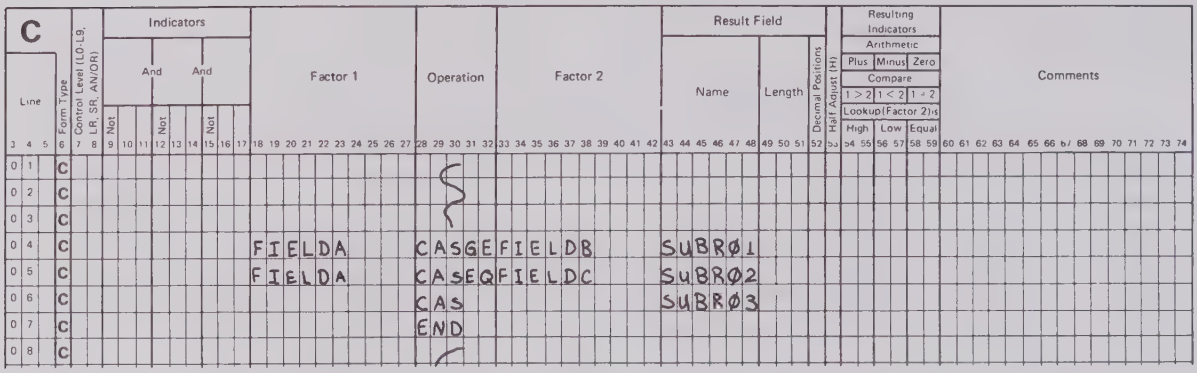
The CASbb operation with no resulting indicators specified in positions 54 through 59 is functionally identical to an EXSR operation, because it causes the unconditional running of the subroutine named in the result field of the CASbb operation. Any CASxx operations that follow an unconditional CASbb operation in the same CAS

group are never tested. Therefore, the normal placement of the unconditional CASbb operation is after all other CASxx operations in the CAS group.

You must not use conditioning indicators on the END operation for a CAS group.

See "Compare Operations" on page 9-9 for further rules for the CASxx operation.

Figure 9-5 shows the coding for a group of CASxx operations.



The CASGE operation (line 04) compares FIELDA with FIELDB. If FIELDA is greater than or equal to FIELDB, SUBR01 is processed and the program continues with the operation specified on line 08. If FIELDA is not greater than or equal to FIELDB, the program next compares FIELDA with FIELDC (line 05). If FIELDA is equal to FIELDC, SUBR02 is processed and the program continues with the operation specified on line 08. If FIELDA is not equal to FIELDC, the CASbb operation (line 06) causes SUBR03 to be processed before the program continues with the operation specified on line 08. The END operation on line 07 denotes the end of the CAS group.

Figure 9-5. CASxx Operation

CHAIN (Random Retrieval from a File Based on Record Number or Key Value)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Optional	R: NR	Opt: ER	Blank

The CHAIN operation retrieves a record from a full procedural file (F in position 16 of the file description specifications), sets a record identifying indicator on (if specified on the input specifications), and places the data from the record into the input fields.

Factor 1, the search argument, must contain the key or relative record number used to retrieve the record. If access is by key, factor 1 can be a field name or a literal. In addition, a KLIST name can be specified in factor 1 for an externally described file. If access is by relative record number, the search argument must be an integer literal or a numeric field with zero decimal positions.

Factor 2 specifies the file or record format name that is to be read. A record format name is valid only with an externally described file. If factor 2 is a file name and access is by key, the CHAIN operation retrieves the first record that matches the search argument.

If factor 2 is a record format name and access is by key, the CHAIN operation retrieves the first record of the specified record type whose key matches the search argument. If no record is found of the specified record type that matches the search argument, a no-record-found condition exists.

A data structure name can be specified in the result field only if the file named in factor 2 is a program described file (identified by an F in position 19 of the file description specification). When a data structure name is specified in the result field, the CHAIN operation retrieves the first record whose record identifier matches the search argument in factor 1 and places it in the data structure. See "File Operations" on page 9-13 for information on how data is transferred between the file and the data structure.

For a WORKSTN file, the CHAIN operation retrieves a subfile record.

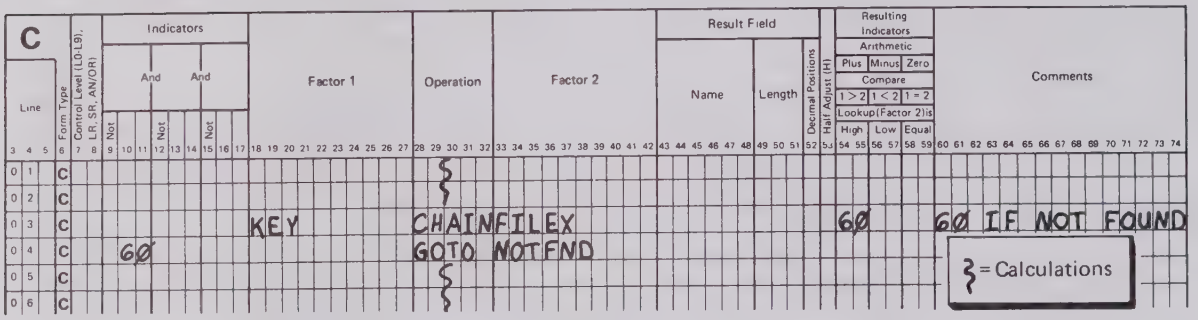
For a multiple device file, you must specify a record format in factor 2. Data is read from the program device identified by the field specified in the ID entry of the file specifications continuation line. If there is no such entry, data is read from the device for the last successful input operation to the file.

Positions 54 and 55 must contain an indicator that is set on if no record in the file matches the search argument. Positions 56 and 57 can contain an indicator to be set on if the CHAIN operation does not complete successfully. Positions 58 and 59 must be blank.

When the CHAIN operation is successful, the file specified in factor 2 is positioned such that a subsequent read operation retrieves the next sequential record following the chained record. When the CHAIN operation does not complete successfully (for example, an error occurs or no record is found), the file specified in factor 2 must be repositioned (for example, by a CHAIN or SETLL operation) before a subsequent read operation can be done on that file.

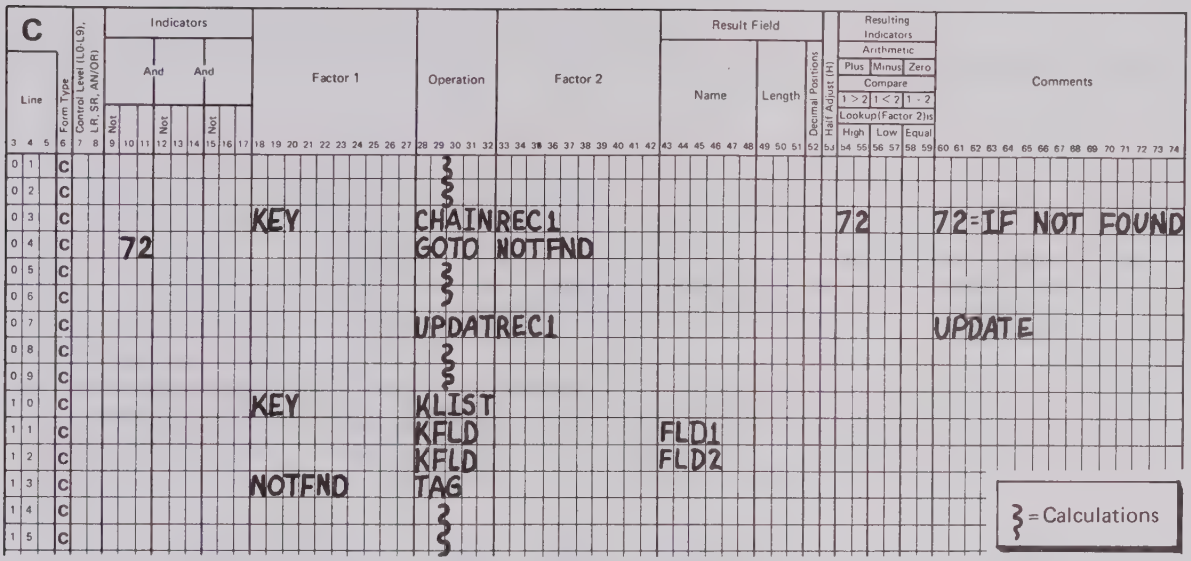
If an update (on the calculation or output specifications) is done on the file specified in factor 2 after a successful CHAIN operation to that file, the last record retrieved is the one updated.

Figure 9-6 and Figure 9-7 on page 9-33 illustrate uses of the CHAIN operation.



The CHAIN operation (line 03) retrieves the first record from the file, FILEX, that has a key field with the same value as the search argument KEY (factor 1). If a record with a key value equal to the search argument is not found, indicator 60 is set on and the GOTO operation conditioned by indicator 60 is processed (line 04). If a record is found with a key value equal to the search argument, the program continues with the calculations at line 05.

Figure 9-6. CHAIN Operation with a File Name in Factor 2



The CHAIN operation (line 03) uses the value contained in the search argument KEY to retrieve a record of the record type REC1 from an externally described file. If no record is found of the specified type that has a key field equal to the search argument, indicator 72 is set on and the program branches to the label specified in the GOTO operation (line 13). If a record of the specified type is found that has a key field equal to the search argument, the calculations starting with line 05 are processed. The UPDAT operation (line 07) modifies all the fields in the REC1 record.

Figure 9-7. CHAIN Operation with a Record Format Name in Factor 2

CLOSE (Close Files)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Opt: ER	Blank

The explicit CLOSE operation closes a file and disconnects the file or device from the program; the file cannot be used again in the program unless an explicit OPEN operation is specified for the file. If a CLOSE operation is issued to files that are not open, no operation is done on the already closed files.

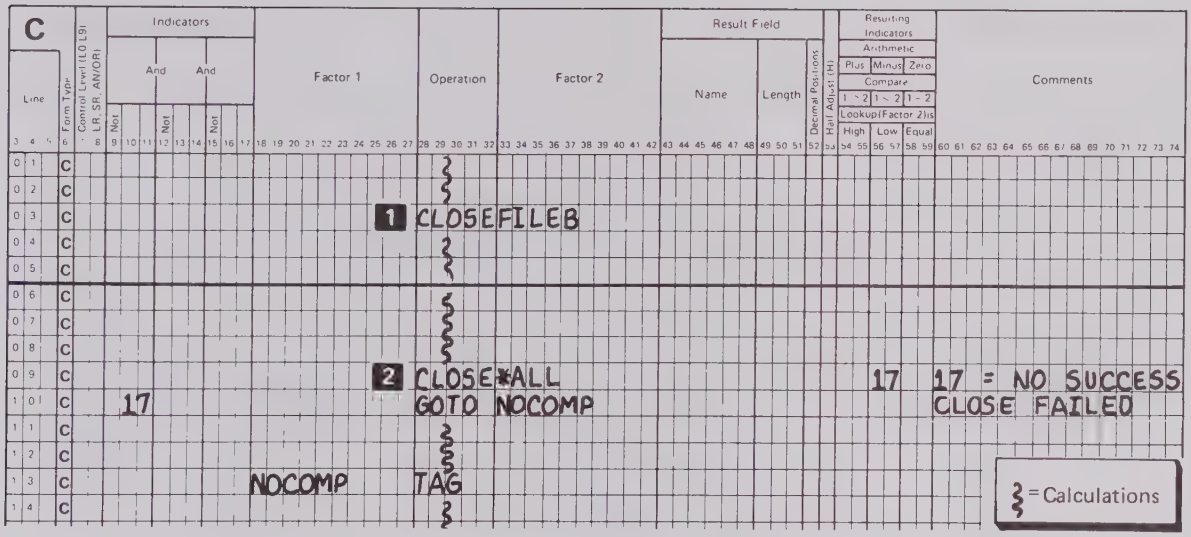
Factor 2 names the file to be closed. The keyword *ALL can be specified in factor 2 to close all the files at once. An array or table file (identified by a T in position 16 of the file description specifications) cannot be specified in factor 2.

A resulting indicator can be specified in positions 56 and 57 to be set on if the CLOSE operation is not completed successfully. Positions 54, 55, 58 and 59 must be blank.

Multiple CLOSE operations to the same file are valid.

If an array or table is to be written to an output file (specified in positions 19 through 26 of the extension specifications), the array or table dump does not occur if the file is closed (by a CLOSE operation) at LR time when RPG/400 writes the file. If the file is closed, it must be reopened in order for the dump to occur.

Figure 9-8 on page 9-35 shows examples of the CLOSE operation.



1 The explicit `CLOSE` operation (line 03) closes `FILEB`.

2 The explicit `CLOSE *ALL` operation (line 09) closes all files in the program. For the programmer to reopen the files, an explicit `OPEN` operation must be specified for each file. If the `CLOSE` operation is not completed successfully, indicator 17 is set on and the program branches to the label specified in the `GOTO` statement (line 13).

Figure 9-8. CLOSE Operations

COMIT (Commit)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Opt	Blank	Blank	Blank	Opt: ER	Blank

The COMIT operation:

- Makes all the changes to your files that have been specified in output operations since the previous COMIT or ROLBK operation (or since the beginning of operations under commitment control if there has been no previous COMIT or ROLBK operation).
- Releases all the record locks for files you have under commitment control.

The file changes and the record-lock releases apply to all the files you have under commitment control, whether the changes have been requested by the program issuing the COMIT operation, or by another program in the same routing step. It is not necessary for the program issuing the COMIT operation to have any files under commitment control.

In factor 1, you may specify a literal, field, array element, table element, data structure, or data structure subfield to identify the boundary between the changes made by this COMIT operation and subsequent changes. At most, the first 4096 bytes of the identifier are used. If you leave factor 1 blank, the identifier is null. On the CL command STRCMTCTL (Start Commitment Control) you may specify an object to which this identifier is written when there is an abnormal job or system end.

The COMIT operation does not change the file position.

The optional indicator in positions 56 and 57 is set on if the operation is not completed successfully. For example, the indicator is set on if commitment control is not active.

COMP (Compare)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Blank	Opt: HI	Opt: LO	Opt: EQ

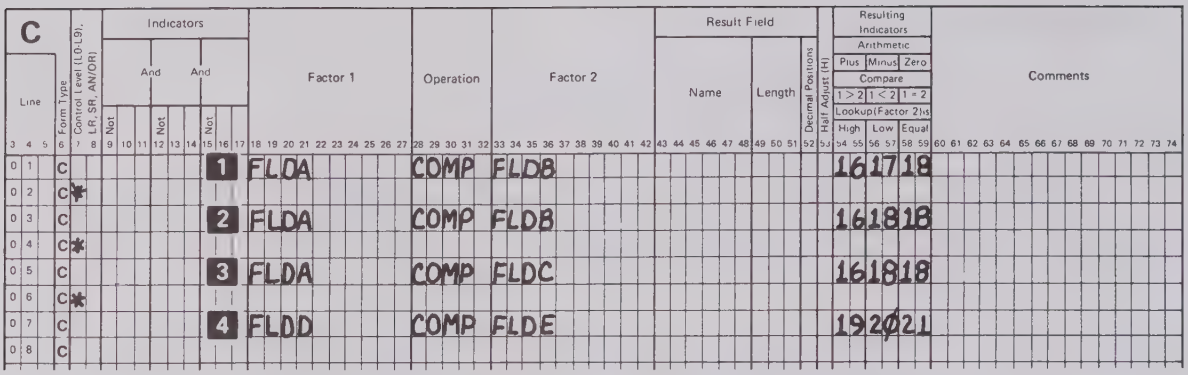
The COMP operation compares factor 1 with factor 2. Both factor 1 and factor 2 must be character data, or both must be numeric. As a result of the compare, indicators are set on as follows:

High Factor 1 is greater than factor 2.
Low Factor 1 is less than factor 2.
Equal Factor 1 equals factor 2.

At least one resulting indicator must be specified in positions 54 through 59. The same indicator cannot be specified for all three conditions (high, low, and equal). When specified, the resulting indicators are set on or off (for each cycle) to reflect the results of the compare.

For further rules for the COMP operation, see "Compare Operations" on page 9-9.

Figure 9-9 on page 9-38 illustrates uses of the COMP operation.



The initial field values are:

```
FLDA = 100.00
FLDB = 105.00
FLDC = -100.00
FLDD = ABC
FLDE = ABCDE
```

As a result of the COMP operations, the resulting indicators are set as follows:

1. Indicator 17 is set on; indicators 16 and 18 are set off.
2. Indicator 18 is set on; indicator 16 is set off.
3. Indicator 16 is set on; indicator 18 is set off.
4. Indicator 20 is set on; indicators 19 and 21 are set off.

Figure 9-9. COMP Operations

DEBUG (Debug Function)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Optional	Optional	Blank	Blank	Blank

The DEBUG operation is an RPG/400 function that helps the programmer debug a program that is not working properly. One or more records containing information helpful for finding programming errors are written as a result of this operation. The DEBUG operation can be used independently of or in combination with the OS/400 testing and debugging functions.

The DEBUG operation code can be specified at any point or at several points in the calculation specifications. Whenever the program encounters the DEBUG operation, one or more records are written, depending on the specifications entered. The first record contains the source statement sequence number of the DEBUG operation in the program, a list of all indicators that are on at the time the DEBUG operation was encountered, and can contain the contents of a field, an array element, a table element, or a literal that identifies the DEBUG written information.

Factor 1 can contain a field name, an array element, a table name, or a literal whose contents identify the DEBUG written information. The length of the specified field can be from 1 to 8 characters. Factor 1 cannot contain a figurative constant. If factor 1 contains an entry, the contents of the entry are written to the first record. If factor 1 is not used, the source statement sequence number of the DEBUG operation is written to the first record.

Factor 2 can contain the name of the output file to which the records are written. This file must have a record length of at least 80 positions. If factor 2 is blank, the output goes to the system console or to the display work station that requested the program (the requester). The same entry must appear in factor 2 of all DEBUG operations in a program.

The result field can contain a field name, an array name, a table name, or a KLIST name whose contents are to be written to a separate record.

If factor 1 and the result field are not specified, only the indicators and the source statement sequence number of the DEBUG operation are written.

Note: If a KLIST is specified in the result field of a DEBUG operation, all numeric fields in the KLIST are printed or displayed in zoned decimal format.

The DEBUG operation is processed only if a 1 is specified in position 15 of the control specification. If the control specification entry is not made, the DEBUG statement is checked for errors at compile time, but the DEBUG operation is not processed at run time.

Records Written for DEBUG

For a DEBUG operation, one record is always written and appears in the following format:

Positions	Information
1-8	DEBUG=
9-16	Source statement sequence number of the DEBUG operation code in the program. This entry permits you to identify the individual DEBUG operation (if more than one is used) without making an entry in factor 1.
17-18	Blank.
19-26	Contents of factor 1, if specified.
27	Minus (-) sign if factor 1 contains a negative value. Blank if factor 1 contains a positive value.
28	Blank.
29-43	The words INDICATORS ON=
44	Blank.
45-47, ...	The names of all indicators that are on, each separated by a blank. If more than one record is needed to write all the indicators, only the first record contains information in positions 1 through 43. The indicators are written starting in position 45 of the remaining records. **Note:** If an indicator contains an invalid character (not 0 or 1), the indicator is listed followed by the hexadecimal representation of the value in the indicator enclosed in parentheses. For example, if indicators 01 and 88 are on, and indicator 33 contains the character A, the indicator line would appear as . . . ON = 01 33 (C1) 88.

When the result field contains an entry, a separate record is written in the following format:

Positions	Information
1-14	The words FIELD VALUE=
15- (any position)	The contents of the result field. The first 66 characters of the result field are written in positions 15 through 80 of this record. If the field is greater than 66 characters, the additional characters are written in positions 15 through 80 of additional records.

Numeric fields are written out in unpacked format and are zero-suppressed. The sign is always written to the right of the field; a minus (-) sign is written to the right of negative fields, and a blank is written to the right of positive fields. Zero fields are written with the last zero and the sign. No other editing is done, If the result

field is an array name, the elements are written in order, one record for each element. When a multiple occurrence data structure is specified, only the current occurrence is written.

DEFN (Field Definition)

Indicators				Result	Resulting Indicators		
7-8	9-17	Factor 1	Factor 2	Field	54-55	56-57	58-59
Opt	Blank	Required	Optional	Required	Blank	Blank	Blank

Depending on the factor 1 entry, the declarative (nonrunnable) DEFN operation can do either of the following:

- Define a field based on the attributes (length and decimal positions) of another field.
- Define a field as a data area.

The DEFN operation can be specified anywhere within calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, the LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry is used for documentation only. Conditioning indicator entries (positions 9 through 17) are not permitted.

*LIKE DEFN

The DEFN operation with *LIKE in factor 1 defines a field based upon the attributes (length and decimal positions) of another field.

Factor 2 must contain the name of the field being referenced, and the result field must contain the name of the field being defined. The field specified in factor 2, which can be defined in the program or in data description specifications (DDS), provides the attributes for the field being defined. Factor 2 cannot be a literal. If factor 2 is an array, an array element, or a table name, the attributes of an element of the array or table are used to define the field. The result field cannot be an array, an array element, or a table name.

Positions 49 through 51 (field length) can be used to make the result field entry longer or shorter than the factor 2 entry. A plus sign (+) in position 49 indicates a length increase; a minus sign (-) in position 49 indicates a length decrease. Positions 50 and 51 can contain the increase or decrease in length (right-adjusted) or can be blank. If positions 49 through 51 are blank, the result field entry is defined with the same length as the factor 2 entry. The number of decimal positions cannot be changed for the field being defined.

See Figure 9-10 on page 9-44 for examples of *LIKE DEFN.

*NAMVAR DEFN

The DEFN operation with *NAMVAR in factor 1 associates a field, a data structure, a data structure subfield, or a data area data structure (within your RPG/400 program) with a AS/400 data area (outside your RPG/400 program).

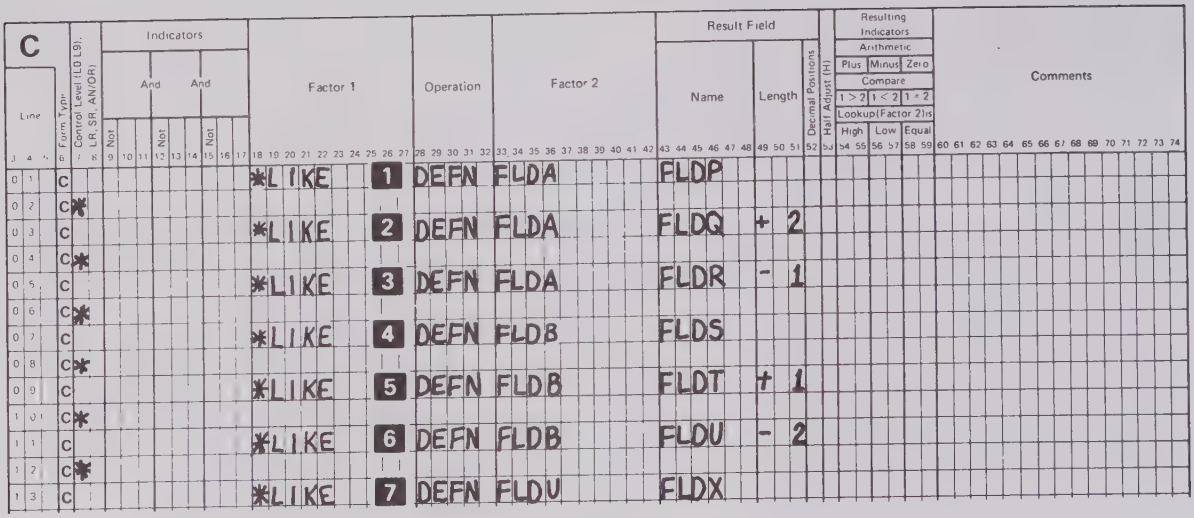
In factor 2, specify the external name of a data area. Use *LDA for the name of the local data area. If you leave factor 2 blank, the result field entry is both the RPG/400 name and the external name of the data area.

In the result field, specify the name of one of the following that you have defined in your program: a field, a data structure, a data structure subfield, or a data area data structure. You use this name with the IN and OUT operations to retrieve data from and write data to the data area specified in factor 2. When you specify a data area data structure in the result field, RPG/400 implicitly retrieves data from the data area at program start and writes data to the data area when the program ends.

The result field entry must not be the name of a file, a program status data structure, a file information data structure (INFDS), a multiple occurrence data structure, an input record field, an array, an array element, or a table. It cannot be the name of a subfield of a multiple occurrence data structure, of a data area data structure, of a program status data structure, of a file information data structure (INFDS), or of a data structure that appears on a *NAMVAR DEFN statement.

In positions 49 through 52, you may define the length and number of decimal positions for the entry in the result field. These specifications must match those for the external description of the data area specified in factor 2. The local data area is character data of length 1024, but within your program, you may access the local data area as if it had a length of 1024 or less.

See Figure 9-10 on page 9-44 for examples of *NAMVAR DEFN.



Line	Form Type	Control Level (L0,L1-L9); or (L, LR, SR, AN/OR)	Indicators: And	Indicators: And	Factor 1	Operation	Factor 2	Result Field: Name	Result Field: Length	Result Field: Decimal Positions	Result Field: Half Adjust (H)	Resulting Indicators: Arithmetic: Plus	Resulting Indicators: Arithmetic: Minus	Resulting Indicators: Arithmetic: Zero	Resulting Indicators: Compare: 1 > 2	Resulting Indicators: Compare: 1 < 2	Resulting Indicators: Compare: 1 = 2	Resulting Indicators: Lookup (Factor 2) is: High	Resulting Indicators: Lookup (Factor 2) is: Low	Resulting Indicators: Lookup (Factor 2) is: Equal	Comments
01	C				*LIKE	1 DEFN	FLDA	FLDP													
02	C	*			*LIKE	2 DEFN	FLDA	FLDQ		+ 2											
03	C				*LIKE	3 DEFN	FLDA	FLDR		- 1											
04	C	*			*LIKE	4 DEFN	FLDB	FLDS													
05	C				*LIKE	5 DEFN	FLDB	FLDT		+ 1											
06	C	*			*LIKE	6 DEFN	FLDB	FLDU		- 2											
07	C				*LIKE	7 DEFN	FLDU	FLDX													
08	C	*																			
09	C																				
10	C	*																			
11	C																				
12	C	*																			
13	C																				

FLDA is a 7-position character field.

FLDB is a 5-digit field with 2 decimal positions.

1 FLDP is a 7-position character field.

2 FLDQ is a 9-position character field.

3 FLDR is a 6-position character field.

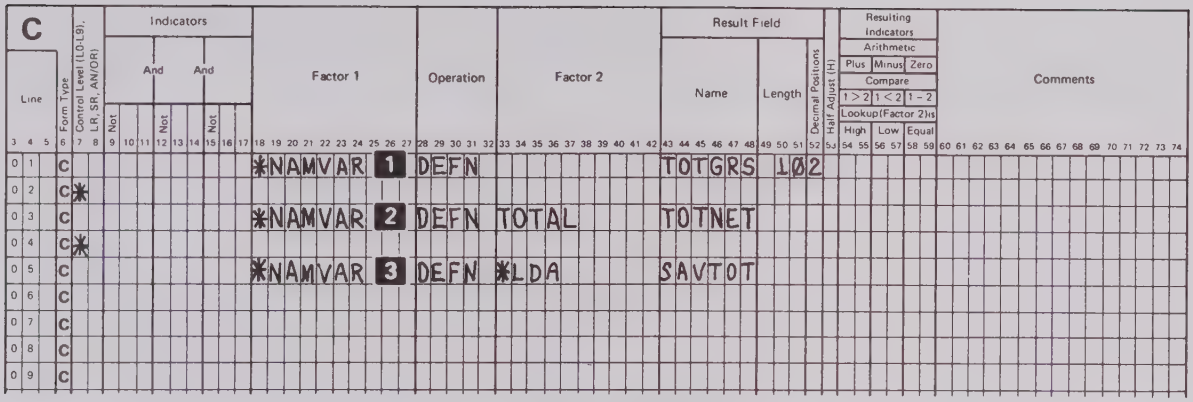
4 FLDS is a 5-position numeric field with 2 decimal positions.

5 FLDT is a 6-position numeric field with 2 decimal positions.

6 FLDU is a 3-position numeric field with 2 decimal positions.

7 FLDX is a 3-position numeric field with 2 decimal positions.

Figure 9-10 (Part 1 of 2). DEFN Operations



| C | | | | Indicators | | | |

DELET (Delete Record)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Required	Blank	Opt: NR	Opt: ER	Blank

The DELET operation deletes a record from a data base file. The file must be an update file (identified by a U in position 15 of the file description specifications). The deleted record can never be retrieved.

Factor 1, the search argument, can contain a key or relative record number that identifies the record to be deleted. If access is by key, factor 1 can be a field name or a literal. In addition, a KLIST name can be specified in factor 1 for an externally described file. If duplicate records exist for the key, only the first of the duplicate records is deleted from the file. If access is by relative record number, the search argument must be an integer literal or a numeric field with zero decimal positions. If factor 1 contains no entry, the DELET operation deletes the current record (the last record retrieved).

Factor 2 must contain the name of the update file or the name of a record format in the file from which a record is to be deleted. A record format name is valid only with an externally described file. The record format name must be the name of the last record read from the file; otherwise an error occurs.

If factor 1 has an entry, you must specify a resulting indicator in positions 54 and 55. If factor 1 does not have an entry, leave these positions blank. This indicator is set on if the record to be deleted is not found in the file. A resulting indicator may be specified in positions 56 and 57; it is set on if the DELET operation does not complete successfully (for example, the DELET operation will not succeed if the user lacks the authority to delete the record). Leave positions 58 and 59 blank.

If a read operation is done on the file specified in factor 2 after a successful DELET operation to that file, the next record after the deleted record is obtained.

DIV (Divide)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Required	Required	Opt: +	Opt: –	Opt: Z

If factor 1 is specified, factor 1 (dividend) is divided by factor 2 (divisor); the quotient (result) is placed in the result field. If factor 1 is 0, the result of the divide operation is 0. Factor 2 cannot be 0. If it is, an error occurs and the RPG/400 exception/error handling routine receives control. When factor 1 is not specified, the result field (dividend) is divided by factor 2 (divisor), and the result (quotient) is placed in the result field.

Any remainder resulting from the divide operation is lost unless the move remainder (MVR) operation is specified as the next operation. If you use conditioning indicators, it is your responsibility to ensure that the DIV operation is processed immediately before the MVR operation. If the MVR operation is processed before the DIV operation, undesirable results occur. For example, error message RPG0907 may be issued. If move remainder is the next operation, the result of the divide operation cannot be half-adjusted (rounded).

For further rules for the DIV operation, see "Arithmetic Operations" on page 9-5.

Figure 9-1 on page 9-7 shows examples of the DIV operation.

DO (Do)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Optional	Optional	Blank	Blank	Blank

The DO operation begins a group of operations you want to process a fixed number of times. An associated END statement marks the end of the group. For further information on do groups, see "Structured Programming Operations" on page 9-11.

On the DO statement, you indicate the number of times the group of operations is to be processed by specifying an index field, a starting value, and a limit value.

In factor 1, specify a starting value with zero decimal positions using a numeric literal or a numeric field name. If you do not specify factor 1, the starting value is 1.

In factor 2, specify a limit value with zero decimal positions using a numeric literal or a numeric field name. If you do not specify factor 2, the limit value is 1.

In the result field, specify a numeric field name that will contain the current index value. If you do not specify an index field, RPG/400 generates one for internal use. Any value in the index field is replaced by factor 1 when the DO operation begins.

In factor 2 of the associated END statement, specify an increment value using a positive numeric literal or numeric field with zero decimal positions. If you leave factor 2 of the associated END statement blank, the increment value of the DO operation is 1.

In addition to the DO operation itself, the conditioning indicators on the DO and END statements control the do group. The conditioning indicators on the DO statement control whether or not the DO operation is begun. The conditioning indicators on the associated END statement control whether or not the do group is repeated another time.

Here is how the DO operation works:

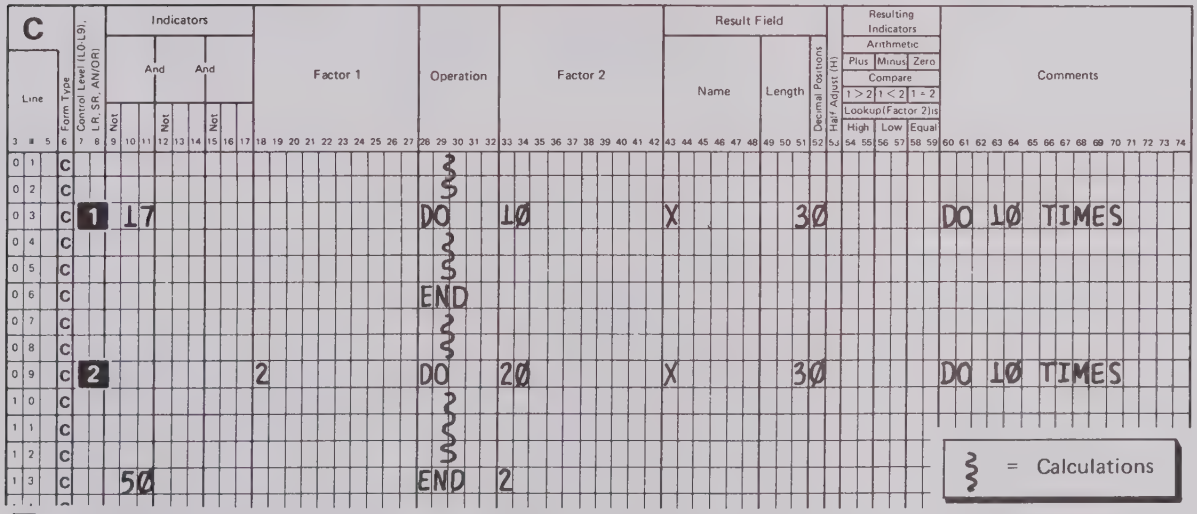
1. If the conditioning indicators on the DO statement line are satisfied, the DO operation is processed (step 2). If the indicators are not satisfied, control passes to the next operation to be processed following the associated END statement (step 7).
2. RPG/400 begins the DO operation by moving the starting value (factor 1) to the index field (result field).
3. If the index value is greater than the limit value, control passes to the calculation operation following the associated END statement (step 7). Otherwise, control passes to the first operation after the DO statement (step 4).
4. Each of the operations in the do group is processed.

5. If the conditioning indicators on the END statement are not satisfied, control passes to the calculation operation following the associated END statement (step 7). Otherwise, the END operation is processed (step 6).
6. RPG/400 processes the END operation by adding the increment to the index field. Control passes to step 3. (Note that the conditioning indicators on the DO statement are not tested again (step 1) when control passes to step 3.)
7. The statement after the END statement is processed when the conditioning indicators on the DO or END statements are not satisfied (steps 1 or 5), or when the index value is greater than the limit value (step 3).

Remember the following when specifying the DO operation:

- The index, increment, limit value, and indicators can be modified within the loop to affect the ending of the do group.
- A do group cannot span both detail and total calculations.

Figure 9-11 illustrates a use of the DO operation.



1 The do group (lines 03 through 06) is processed 10 times when indicator 17 is on; it stops running when the index value in field X, the result field, is greater than the limit value (10) in factor 2. When the do group stops running, control passes to the operation immediately following the END operation (line 07). Because factor 1 in the DO operation is not specified, the starting value is 1. Because factor 2 of the END operation is not specified, the incrementing value is 1.

2 The do group (lines 09 through 13) can be processed 10 times. The do group stops running when the index value in field X is greater than the limit value (20) in factor 2, or if indicator 50 is not on when the END operation is encountered. When indicator 50 is not on, the END operation is not processed; therefore, control passes to the operation following the END operation. The starting value of 2 is specified in factor 1 of the DO operation, and the incrementing value of 2 is specified in factor 2 of the END operation.

Figure 9-11. DO Operation

DOUxx (Do Until)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Blank	Blank	Blank	Blank

The DOUxx operation begins a group of operations you want to process more than once (but always at least once). An associated END statement marks the end of the group. For further information on do groups and the meaning of xx, see "Structured Programming Operations" on page 9-11.

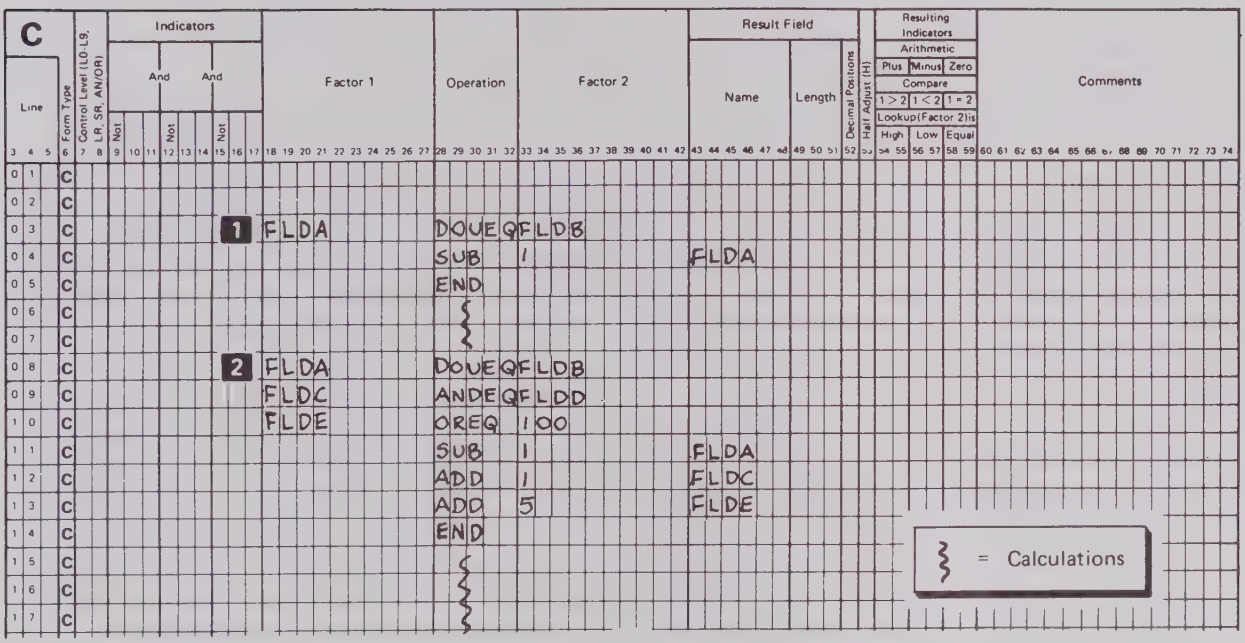
On the DOUxx statement, you indicate a relationship xx. You may also specify a more complex condition by immediately following the DOUxx statement statement with ANDxx or ORxx statements. The operations in the do group are processed once, and then the group is repeated until the relationship exists between factor 1 and factor 2 or the condition specified by a combined DOUxx, ANDxx, or ORxx operation exists.

In addition to the DOUxx operation itself, the conditioning indicators on the DOUxx and END statements control the do group. The conditioning indicators on the DOUxx statement control whether or not the DOUxx operation is begun. The conditioning indicators on the associated END statement control whether or not the do group is repeated another time.

Here is how the DOUxx operation works:

1. If the conditioning indicators on the DOUxx statement line are satisfied, the DOUxx operation is processed (step 2). If the indicators are not satisfied, control passes to the next operation that can be processed following the associated END statement (step 6).
2. RPG/400 processes the DOUxx operation by passing control to the next operation that can be processed (step 3). The DOUxx operation does not compare factor 1 and factor 2 or test the specified condition at this point.
3. Each of the operations in the do group is processed.
4. If the conditioning indicators on the END statement are not satisfied, control passes to the next calculation operation following the associated END statement (step 6). Otherwise, the END operation is processed (step 5).
5. RPG/400 processes the END operation by comparing factor 1 and factor 2 of the DOUxx or testing the condition specified by a combined operation. If the relationship xx exists between factor 1 and factor 2 or the specified condition exists, the do group is finished and control passes to the next calculation operation after the END statement (step 6). If the relationship xx does not exist between factor 1 and factor 2 or the specified condition does not exist, the operations in the do group are repeated (step 3).

6. The statement after the END statement is processed when the conditioning indicators on the DOUxx or END statements are not satisfied (steps 1 or 4), or when the relationship xx between factor 1 and factor 2 or the specified condition exists at step 5.



1 The DOUEQ operation runs the operation within the do group (line 04) at least once. At the END operation (line 05), the test is processed to determine whether FLDA is equal to FLDB. If FLDA does not equal FLDB, line 04 is processed again. This loop continues processing until FLDA is equal to FLDB. When FLDA is equal to FLDB, the program branches to the operation immediately following the END operation (line 06).

2 The combined DOUEQ ANDEQ OREQ operation processes the operation within the do group (lines 11 to 13) at least once. At the END operation (line 14), a test is processed to determine whether the specified condition, FLDA equal to FLDB and FLDC equal to FLDD, exists. If the condition exists, the program branches to the operation immediately following the END operation (line 15). There is no need to test the OREQ condition, FLDE equal to 100, if the DOUEQ and ANDEQ conditions are met. If the specified condition does not exist, the OREQ condition is tested. If the OREQ condition is met, the program branches to the operation immediately following the END. Otherwise, lines 11 through 13 are processed and then the program processes the conditional tests starting at line 8 again. If neither the DOUEQ and ANDEQ condition nor the OREQ condition are met, lines 11 through 13 are processed again.

Figure 9-12. DOUxx Operation

DOWxx (Do While)

Indicators				Result	Resulting Indicators		
7-8	9-17	Factor 1	Factor 2	Field	54-55	56-57	58-59
Opt	Opt	Required	Required	Blank	Blank	Blank	Blank

The DOWxx operation begins a group of operations you want to process while the relationship xx exists between factor 1 and factor 2. You may also specify a more complex condition by immediately following the DOWxx statement with ANDxx or ORxx statements. An associated END statement marks the end of the group. For further information on do groups and the meaning of xx, see "Structured Programming Operations" on page 9-11.

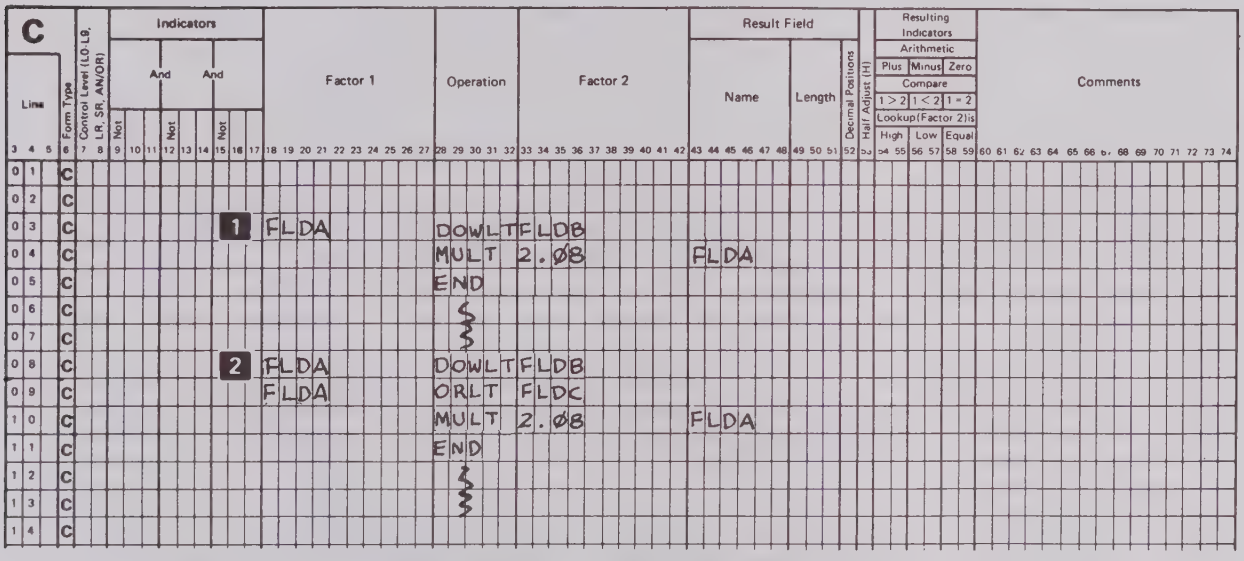
Factor 1 and factor 2 must contain a character literal, a numeric literal, or a field name. Factor 1 and factor 2 must be either both character data or both numeric data. The comparison of factor 1 and factor 2 follows the same rules as those given for the compare operations. See "Compare Operations" on page 9-9.

In addition to the DOWxx operation itself, the conditioning indicators on the DOWxx and END statements control the do group. The conditioning indicators on the DOWxx statement control whether or not the DOWxx operation is begun. The conditioning indicators on the associated END statement control whether or not the do group is repeated another time.

Here is how the DOWxx operation works:

1. If the conditioning indicators on the DOWxx statement line are satisfied, the DOWxx operation is processed (step 2). If the indicators are not satisfied, control passes to the next operation to be processed following the associated END statement (step 6).
2. RPG/400 processes the DOWxx operation by comparing factor 1 and factor 2 or testing the condition specified by a combined DOWxx, ANDxx, or ORxx operation. If the relationship xx between factor 1 and factor 2 or the condition specified by a combined operation does not exist, the do group is finished and control passes to the next calculation operation after the END statement (step 6). If the relationship xx between factor 1 and factor 2 or the condition specified by a combined operation exists, the operations in the do group are repeated (step 3).
3. Each of the operations in the do group is processed.
4. If the conditioning indicators on the END statement are not satisfied, control passes to the next operation to run following the associated END statement (step 6). Otherwise, the END operation is processed (step 5).
5. RPG/400 processes the END operation by passing control to the DOWxx operation (step 2). (Note that the conditioning indicators on the DOWxx statement are not tested again at step 1.)

6. The statement after the END statement is processed when the conditioning indicators on the DOWxx or END statements are not satisfied (steps 1 or 4), or when the relationship xx between factor 1 and factor 2 or the specified condition does not exist at step 2.



1 The DOWLT operation (line 03) allows the operation within the do group (line 04) to be processed only if FLDA is less than FLDB. If FLDA is not less than FLDB, the program branches to the operation immediately following the END operation (line 06). If FLDA is less than FLDB, line 04 is processed. The END operation (line 05) causes the program to branch to line 03 where a test is again made to determine whether FLDA is less than FLDB. This loop continues processing until FLDA is equal to or greater than FLDB; then the program branches to the operation immediately following the END operation (line 06).

2 In this example multiple conditions are tested. The combined DOWLT ORLT operation (lines 08 and 09) allows the operation within the do group (line 10) to be processed only while FLDA is less than FLDB or FLDC. If neither specified condition exists, the program branches to the operation immediately following the END operation (line 12). If either of the specified conditions exist, line 10 is processed. The END operation (line 11) causes the program to branch to line 08 where a test is again made to determine whether specified conditions exist. This loop continues until FLDA is equal to or greater than FLDB and FLDC; then the program branches to the operation immediately following the END operation (line 12).

Figure 9-13. DOWxx Operation

DSPLY (Display Function)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Optional	Optional	Blank	Opt: ER	Blank

The DSPLY operation allows the program to communicate with the system console or with the display work station that requested the program. The operation can display a message and accept a response.

Factor 1 is optional. If specified, factor 1 can contain a field name, a literal, a table name, or an array element whose value is used to create the message to be displayed, or factor 1 can contain *M, followed by a message identifier that identifies the message to be retrieved from the message file, QUSERMSG. QUSERMSG must be in a library that is specified in the library list of the job receiving the message.

The message identifier can be from 1 to 7 characters in length and can include an optional alphabetic prefix followed immediately by 1 to 4 numeric digits. If specified, the alphabetic prefix must be 3 characters in length. If the alphabetic prefix is not specified, the default is USR. The numeric portion of the message identifier (1) must immediately follow either *M or the optional alphabetic prefix, (2) cannot include embedded blanks, and (3) must be left-adjusted (high-order zeros can be omitted). If the numeric portion of the message identifier is not specified, the default is 0000. Valid examples are:

Factor 1 Entry	Message Identifier Used
*M	USR0000
*MABC	ABC0000
*M1	USR0001
*MABC5	ABC0005

If specified, factor 2 can contain a character field, a literal, a table name, or an array element whose value is the symbolic name of the object to receive the message and from which the optional response can be sent. Any queue name except a program message queue name can be the value contained in the factor 2 entry. The queue must be declared to OS/400 before it can be specified in factor 2. (For information on how to create a queue, see the *Programming: Control Language Programmer's Guide*.) There are two predefined queues:

Queue	Value
QSYSOPR	The message is sent to the system operator. Note that the QSYSOPR message queue severity level must be zero (00) to enable the DSPLY operation to immediately display a message to the system operator.
*EXT	The message is sent to the requester.

Note: For a batch job, if factor 2 is blank, the default is QSYSOPR. For an interactive job, if factor 2 is blank, the default is *EXT.

The result field is optional. If specified, the result field can contain a field name, a table name, or an array element in which the response is placed. If you do not respond by entering data, the result field is unchanged. Press Enter to continue.

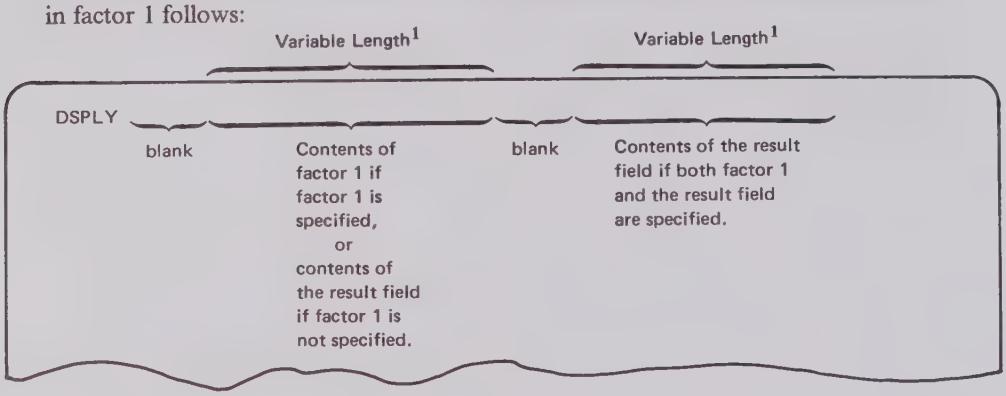
Positions 56 and 57 can contain an indicator to be set on if an error occurs on the operation. If an error is made while the work station user enters the response, the message is redisplayed a maximum of five times. After the fifth display, the indicator in positions 56 and 57, if specified, is set on. If the indicator in positions 56 and 57 is not specified, the exception/error handling routine receives control.

When specifying the DSPLY operation *with no message identifier in factor 1*, the operation functions as follows:

- When factor 1 contains an entry and the result field is blank, the contents of the factor 1 entry are displayed. The program does not wait for a response except in the following case. If you have used a display file to display a format at your work station, and the display file has the parameter RSTDSP (*NO) specified, then the program waits for you to press Enter.
- When factor 1 is blank and the result field contains an entry, the contents of the result field entry are displayed and the program waits for the user to enter data for the response. The reply is placed in the result field.
- When both factor 1 and the result field contain entries, the contents of the factor 1 and result field entries are combined and displayed. The program waits for the user to enter data for the response. The response is placed in the result field.

  The maximum length of information that can be displayed is 52 characters.

The format of the record written by the DSPLY operation with no message identifier in factor 1 follows:



When specifying the DSPLY operation *with a message identifier in factor 1*, the operation functions as follows: the message identified by the factor 1 entry is retrieved from QUSERMSG, the message is displayed, and the program waits for the user to respond by entering data if the result field is specified. The response is placed in the result field. For information on how to format the display, see the *Programming: Data Management Guide*.

When replying to a message, remember the following:

- Numeric fields sent to the display are right-adjusted and zero-suppressed.
- If a numeric field is keyed with a length greater than the number of digits in the result field and the rightmost character is not a minus sign (-), an error is detected and a second wait occurs. The user must rekey the field.
- To enter a null response to the system operator queue (QSYSOPR), the user must enter the characters *N and then press Enter.
- Character fields are padded on the right with blanks after all characters are keyed.
- Numeric fields are right-adjusted and padded on the left with zeros after all characters are keyed.
- Lowercase characters are not converted to uppercase.

[1] The maximum length of information that can be displayed is 52 characters.

DUMP (Program Dump)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Blank	Blank	Blank	Blank	Blank

The DUMP operation provides a dump (all fields, indicators, data structures, arrays, and tables defined) of the program. The DUMP operation can be used independently of or in combination with the OS/400 testing and debugging functions.

Factor 1 can contain a character field name, a literal, a table element, or an array element whose contents identify the dump. Factor 1 cannot contain a figurative constant.

The program continues processing the next calculation statement following the DUMP operation.

The DUMP operation can be used in the calculation specifications only if a 1 is specified in position 15 of the control specification. If the control specification entry is not made, the DUMP operation is checked for errors and the statement is printed on the listing, but the DUMP operation is not processed.

If you have specified a POST operation code, with factor 1 blank, anywhere in your program, RPG/400 does not update any of the file information data structures (INFDS) until you do an appropriate POST operation. For up-to-date information, do a POST operation for each file before you do the DUMP operation. (This action is not required for a dump that the system does in response to an inquiry message.)

ELSE (Else Do)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Blank	Blank	Blank	Blank	Blank	Blank	Blank

The ELSE operation is optional with the IFxx operation. ELSE is specified immediately following the calculations that are processed if the IFxx comparison is met. ELSE is immediately followed by the calculations to be processed if the IFxx comparison is not met.

The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry is for documentation purposes only. Conditioning indicator entries (positions 9 through 17) are not permitted.

An END operation must be used to close the IFxx/ELSE group.

Figure 9-20 on page 9-76 shows an example of an ELSE operation with an IFxx operation.

END (End)

Indicators				Result	Resulting Indicators		
7-8	9-17	Factor 1	Factor 2	Field	54-55	56-57	58-59
Opt	Opt	Blank	Optional	Blank	Blank	Blank	Blank

The END operation specifies the end of a CASxx, DO, DOUxx, DOWxx, or IFxx group of operations.

A factor 2 entry is allowed only on an END operation that delimits a group of statements controlled by a DO operation. In this case, the factor 2 contains the incrementing value of the DO operation and can be a positive numeric literal or a numeric field with zero decimal positions. If factor 2 is not specified in such an END statement, the incrementing value of the DO operation defaults to 1.

For an explanation of how conditioning indicators affect the END operation, see the DO, DOUxx, and DOWxx operation codes in this chapter. Do not use conditioning indicators on the END operation for CASxx and IFxx operations.

ENDSR (End Subroutine)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Blank	Optional	Optional	Blank	Blank	Blank	Blank

The ENDSR operation defines the end of an RPG/400 subroutine and the exit point to the main program. ENDSR must be the last statement in the subroutine. Factor 1 can contain a label that can be used as a point to which a GOTO operation within the subroutine can branch. The control level entry (positions 7 and 8) can be SR or remain blank. Conditioning indicator entries are not permitted.

The ENDSR operation ends a subroutine and causes a branch back to the statement immediately following the EXSR operation unless the subroutine is a program exception/error subroutine (*PSSR) or a file exception/error subroutine (INFSR). For these subroutines, factor 2 of the ENDSR operation can contain an entry that specifies where control is to be returned following processing of the subroutine. This entry can be a field name that contains a reserved keyword or a literal that is a keyword. If an invalid return point is specified, the RPG/400 default error handler receives control.

See Figure 9-17 on page 9-68 for an example of coding an RPG/400 subroutine.

EXCPT (Calculation Time Output)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Optional	Blank	Blank	Blank	Blank

The EXCPT operation has two major functions:

- It allows records to be written during calculation time.
- It allows a variable number of records to be written in one program cycle at either detail calculation or total calculation time.

See Figure 9-14 on page 9-62 through Figure 9-16 on page 9-64 for examples of the EXCPT operation.

Remember the following when specifying the EXCPT operation:

- The exception records that are to be written during calculation time are indicated by an E in position 15 of the output specifications. An EXCPT name, which is the same name as specified in factor 2 of an EXCPT operation, can be specified in positions 32 through 37 of the exception records.
- Only exception records, not heading, detail, or total records, can contain an EXCPT name.
- When the EXCPT operation with a name in factor 2 is processed, only those exception records with the same EXCPT name are checked and written if the conditioning indicators are satisfied.
- When factor 2 is blank, only those exception records with no name in positions 32 through 37 are checked and written if the conditioning indicators are satisfied.
- If an exception record is conditioned by an overflow indicator, the record is written only during the overflow portion of the RPG/400 cycle or during fetch overflow. The record is not written at the time the EXCPT operation is processed.

EXCPT (Calculation Time Output)

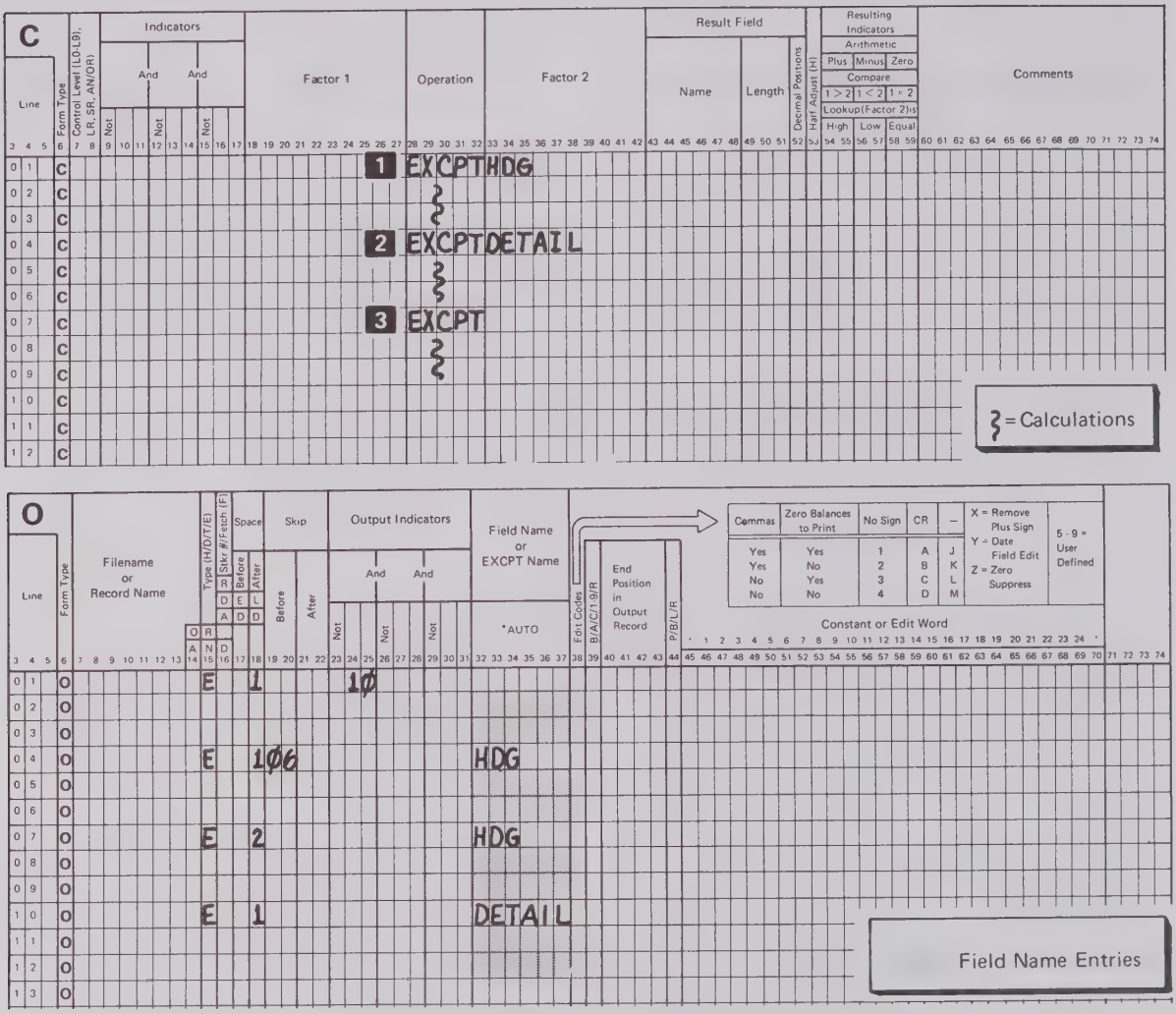


Figure 9-14. EXCPT Operation with/without Factor 2 Specified

1 When the EXCPT operation with HDG specified in factor 2 is processed (line 01 in the calculation specifications), all exception records with the EXCPT name HDG (lines 04 and 07 of the output specifications) are written.

2 When the EXCPT operation with DETAIL specified in factor 2 is processed (line 04 of the calculation specifications), all exception records with the EXCPT name DETAIL (line 10 of the output specifications) are written.

3 When the EXCPT operation with no entry in factor 2 is processed (line 07 of the calculation specifications), all exception records that do not have an EXCPT name specified in positions 32 through 37 (such as line 01 of the output specifications) are written if the conditioning indicators are satisfied. Any exception records without conditioning indicators and without an EXCPT name are always written by an EXCPT operation with no entry in factor 2.

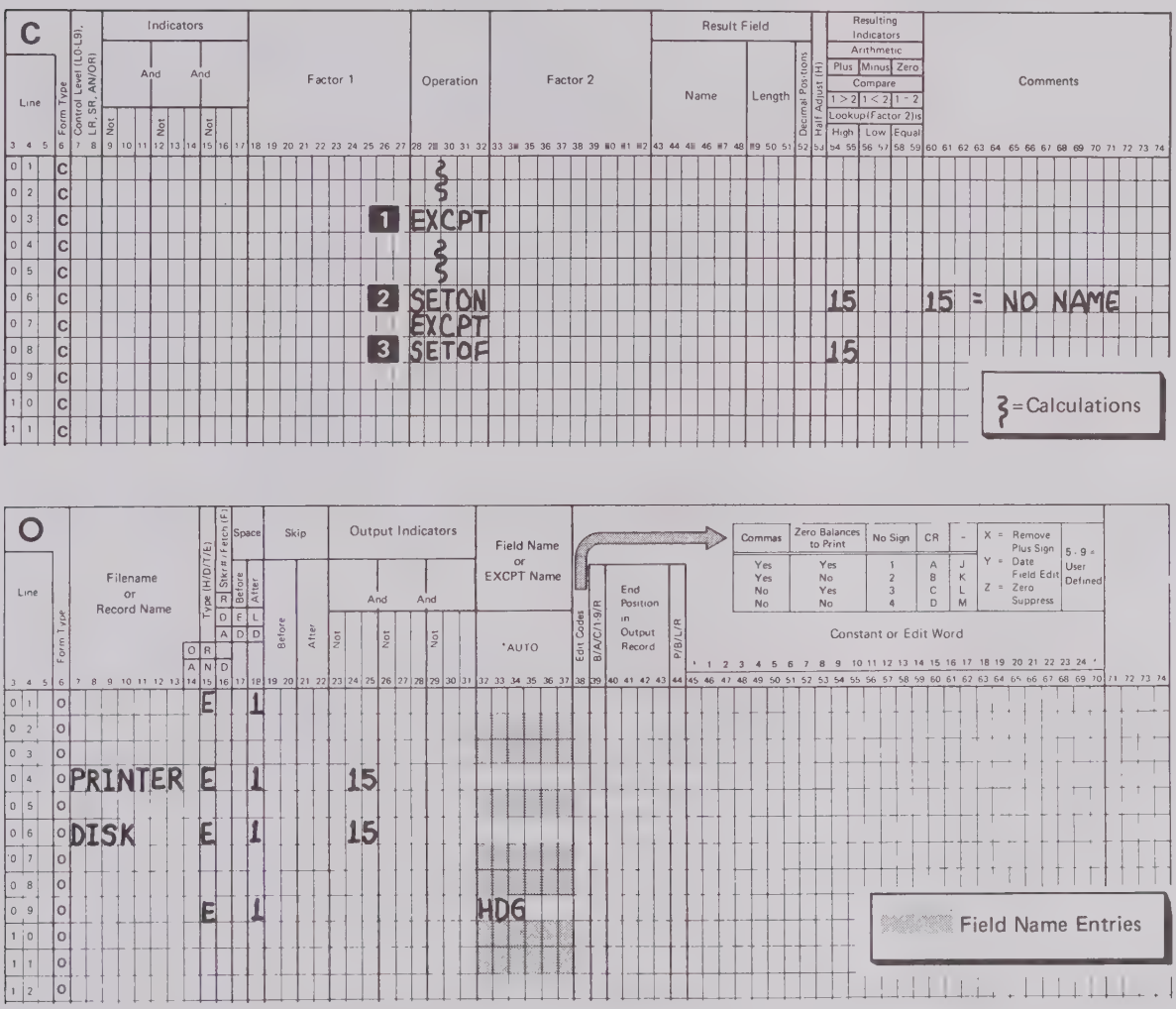


Figure 9-15. EXCPT Operation without Factor 2 Specified

1 When the EXCPT operation is encountered in the calculation specifications (line 03), all exception records in the output specifications that do not have an EXCPT name are written if the conditioning indicators allow output. Line 01 of the output specifications is always output by the EXCPT operation because it is unconditioned.

2 When the SETON operation sets on indicator 15 in the calculation specifications (line 06) and the EXCPT operation on line 07 is processed, all lines in the output specifications that have no EXCPT name in positions 32 through 37 and that are unconditioned or conditioned by indicator 15 (lines 01, 04, and 06) are written.

3 The SETOF operation sets off indicator 15 (line 08).

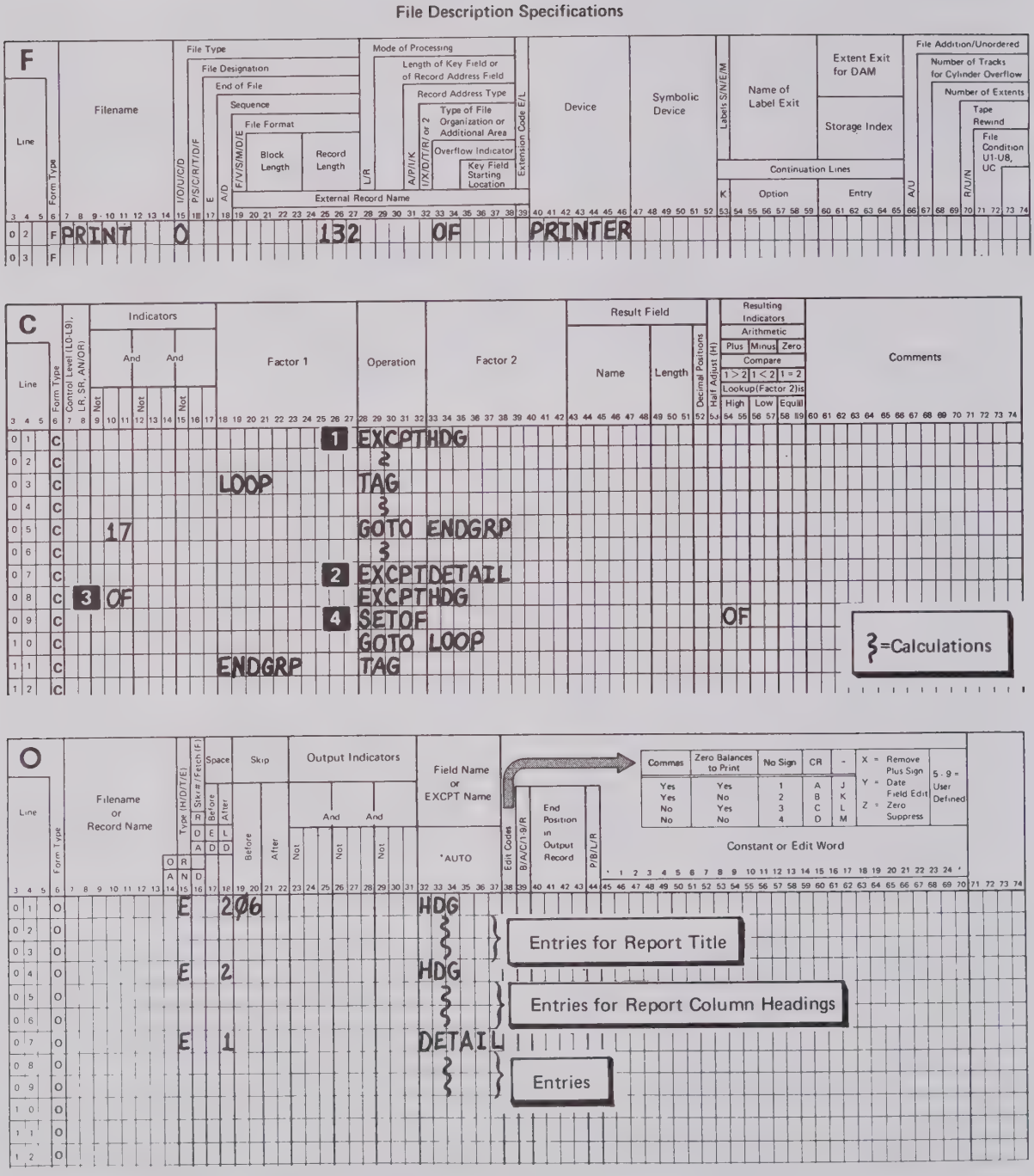


Figure 9-16. EXCPT Output with an Overflow Indicator

This example shows the coding for EXCPT output with an overflow indicator when you are printing the title and column headings on each page of a report.

1 The EXCPT operation with HDG specified in factor 2 (line 01 of the calculation specifications) causes all lines with the group EXCPT name HDG (lines 01 and 04 of the output specifications) to be printed.

2 The EXCPT operation with DETAIL specified in factor 2 (line 07 of the calculation specifications) causes all lines with the group EXCPT name DETAIL (line 07 of the output specifications) to be printed.

3 When the overflow indicator is set on, the EXCPT operation (line 08 of the calculation specifications) prints all HDG lines (lines 01 and 04 of the output specifications) on the overflow page.

4 The SETOF operation sets off the 0F indicator (line 09 of the calculation specifications), and the program branches to the label specified in the GOTO operation.

EXFMT (Execute Format)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Opt: ER	Blank

The EXFMT operation is a combination of a WRITE followed by a READ to the same record format. EXFMT is valid only for a WORKSTN file defined as a full procedural (F in position 16 of the file description specifications) combined file (C in position 15 of the file description specifications) that is externally described (E in position 19 of the file description specifications).

Factor 2 must contain the name of the record format to be written and then read. A resulting indicator can be specified in positions 56 and 57 to be set on if the EXFMT operation does not complete successfully. When the indicator is set on, the read portion of the operation is not processed (record identifying indicators and fields are not modified). Positions 54, 55, 58, and 59 must be blank.

For the use of EXFMT with multiple device files, see the descriptions of the READ (by format name) and WRITE operations, in this chapter.

EXSR (Execute Subroutine)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Blank	Blank

The EXSR operation causes the RPG/400 subroutine named in factor 2 to be processed. The subroutine name must be a unique symbolic name and must appear in factor 1 of a BEGSR operation. The EXSR operation can appear anywhere in the program. Whenever it appears, the subroutine that is named is processed. After operations in the subroutine are processed, the statement following the EXSR operation is processed except when a GOTO within the subroutine is given to a label outside the subroutine or when the subroutine is an exception/error subroutine with an entry in factor 2.

*PSSR used in factor 2 specifies the subroutine as the program exception/error subroutine that handles program-detected errors.

Coding Subroutines

An RPG/400 subroutine can be processed from any point in the calculation operations. All RPG/400 operations can be processed within a subroutine, and these operations can be conditioned by any valid indicators in positions 9 through 17. SR or blanks can appear in positions 7 and 8. Control level indicators (L1 through L9) cannot be used in these positions. However, AND/OR lines within the subroutine can be indicated in positions 7 and 8.

Fields used in a subroutine can be defined either in the subroutine or in the rest of the program. In either instance, the fields can be used by both the main program and the subroutine.

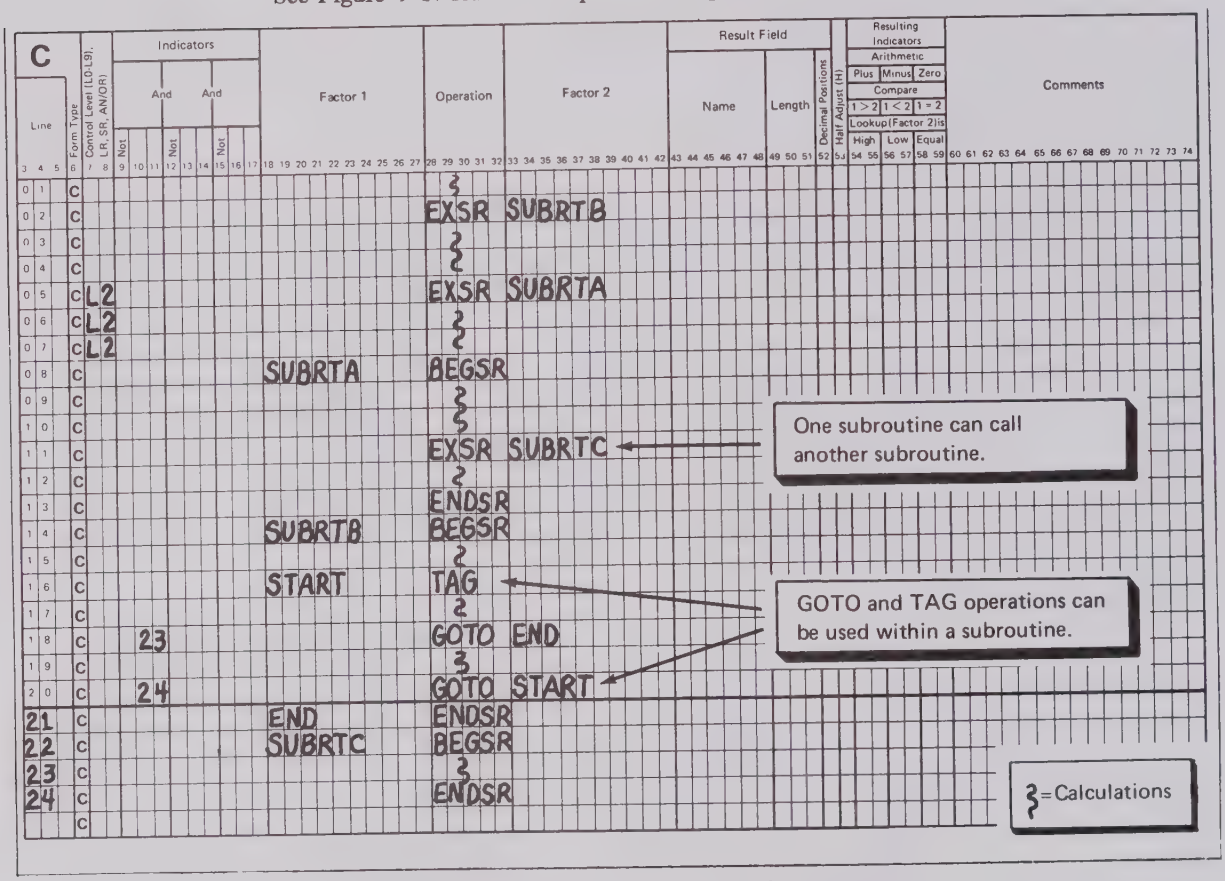
A maximum of 254 subroutines can be included in a program; however, a subroutine cannot contain another subroutine. One subroutine can call another subroutine; that is, a subroutine can contain an EXSR or CASxx. However, an EXSR or CASxx specification within a subroutine cannot directly call itself. Indirect calls to itself through another subroutine should not be performed, since unpredictable results can occur. Use the GOTO and TAG operation codes if you want to branch to another point within the same subroutine.

Subroutines do not have to be specified in the order in which they are used. Each subroutine must have a unique symbolic name. Each subroutine must contain a BEGSR and an ENDSR statement.

The use of the GOTO (branching) operation is permitted within a subroutine. GOTO can specify the label on the ENDSR operation associated with that subroutine; it cannot specify the name of a BEGSR operation. A GOTO outside the subroutine cannot be

issued to a BEGSR, ENDSR, or TAG within a subroutine. A GOTO within a subroutine can be issued to a TAG within either detail or total calculations.

See Figure 9-17 for an example of coding a subroutine.



Note: For a subroutine, positions 7 and 8 can be blank or can contain SR.

Figure 9-17. Example of Coding Subroutines

FEOD (Force End of Data)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Opt: ER	Blank

The FEOD operation signals the logical end of data for a primary, secondary, or full procedural file. The FEOD function differs, depending on the file type and device. (For an explanation of how FEOD differs per file type and device, see the *Programming: Data Base Guide*.)

FEOD differs from the CLOSE operation in that the program is not disconnected from the device or file; the file can be used again for subsequent file operations without an explicit OPEN operation being specified to the file.

Conditioning indicators can be specified. Factor 2 names the file to which FEOD is specified. A resulting indicator can be specified in positions 56 and 57 to be set on if the operation does not complete successfully.

In order to process any further sequential operations to the file after the FEOD operation (for example, READ or READP), the file must be repositioned.

FORCE (Force A Certain File to Be Read on the Next Cycle)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Blank	Blank

The FORCE operation allows selection of the file from which the next record is to be read. The FORCE operation can be used only for primary or secondary files.

Factor 2 must contain the name of a file from which the next record is to be selected.

If the FORCE operation is processed, the record is read at the start of the next program cycle. If more than one FORCE operation is processed during the same program cycle, all but the last is ignored. FORCE must be issued at *detail* time, not total time.

FORCE operations override the multifile processing method by which the program normally selects records. However, the first record to be processed is always selected by the normal method. The remaining records can be selected by FORCE operations.

If FORCE is specified for a file that is at end of file, no record is retrieved from the file. The program cycle determines the next record to be read.

FREE (Deactivate a Program)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Opt: ER	Blank

The FREE operation removes a program from the list of activated programs, frees static storage, and ensures program initialization (first cycle processing) the next time the program is called. It does not close files or unlock data areas.

Factor 2 must contain the name of a field, a literal, or an array element that contains the name of the program to be deactivated. The entry in factor 2 must be character data and may include a qualified name: for example, LIB/PGM. RPG/400 uses the program name exactly as specified in the literal, field, or array element to determine the program to be called. (Lowercase characters are not shifted to uppercase, and a name enclosed in quotation marks (for example "ABC") always includes the quotation marks as part of the name of the program to be freed.)

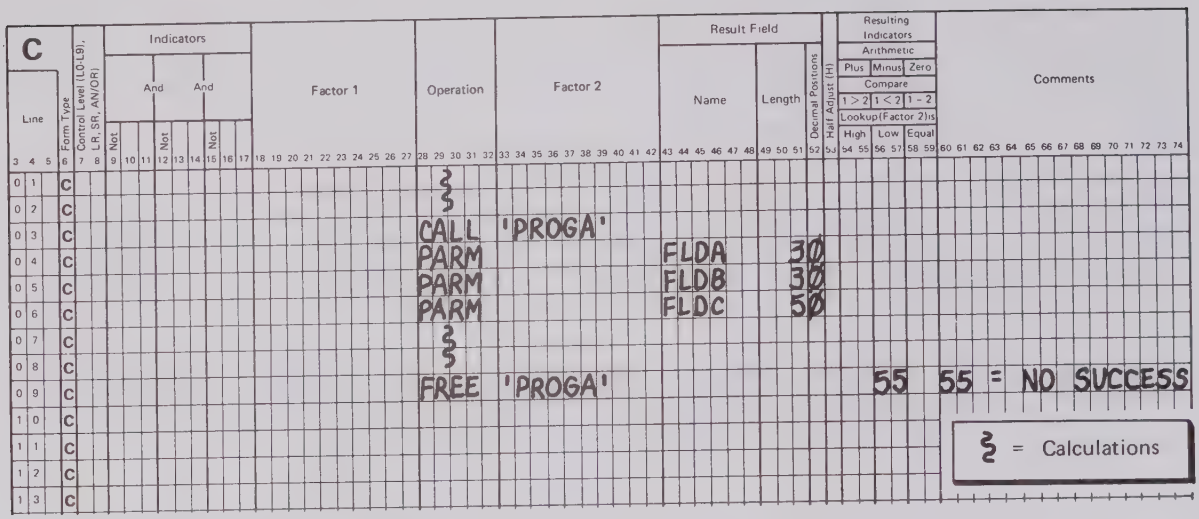
Any valid resulting indicator can be specified in positions 56 and 57 to be set on if FREE is not completed successfully.

For programs that are to run under System/38 Environment, if you specify the library name, the program name must be immediately followed by a period and then the library name.

Note: Issuing a FREE operation and then a CALL operation to the same program reopens the program's files and may use additional temporary storage. Repeatedly issuing FREE and CALL operations to a program, without closing the program's files in between, may use enough temporary storage to degrade the system's performance, and ultimately cause a AS/400 machine check.

Figure 9-18 on page 9-72 shows the FREE operation being used with the CALL operation.

FREE (Deactivate a Program)



When the CALL operation is processed (line 03), the data in the result fields of the parameter list can be accessed by PROGA. The parameter list ends when the first calculation operation other than a PARM operation is encountered (line 07). When the FREE operation is processed (line 09), PROGA is removed from the list of activated programs. Removing PROGA from the list ensures a fresh copy of all fields in PROGA the next time the program is called. Indicator 55 is set on if the FREE operation is not completed successfully.

Figure 9-18. CALL/FREE Operations

GOTO (Branch To)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Blank	Blank

The GOTO operation allows calculation operations to be skipped by instructing the program to go to (or branch to) another calculation operation in the program. A TAG operation names the destination of a GOTO operation. A GOTO operation can be used to specify a branch:

- To a previous or a succeeding specification line
- From a detail calculation line to another detail calculation line
- From a total calculation line to another total calculation line
- From a detail calculation line to a total calculation line
- From a subroutine to a detail calculation line or to a total calculation line.

A GOTO operation outside a subroutine cannot specify a branch to a TAG or ENDSR operation within that subroutine.

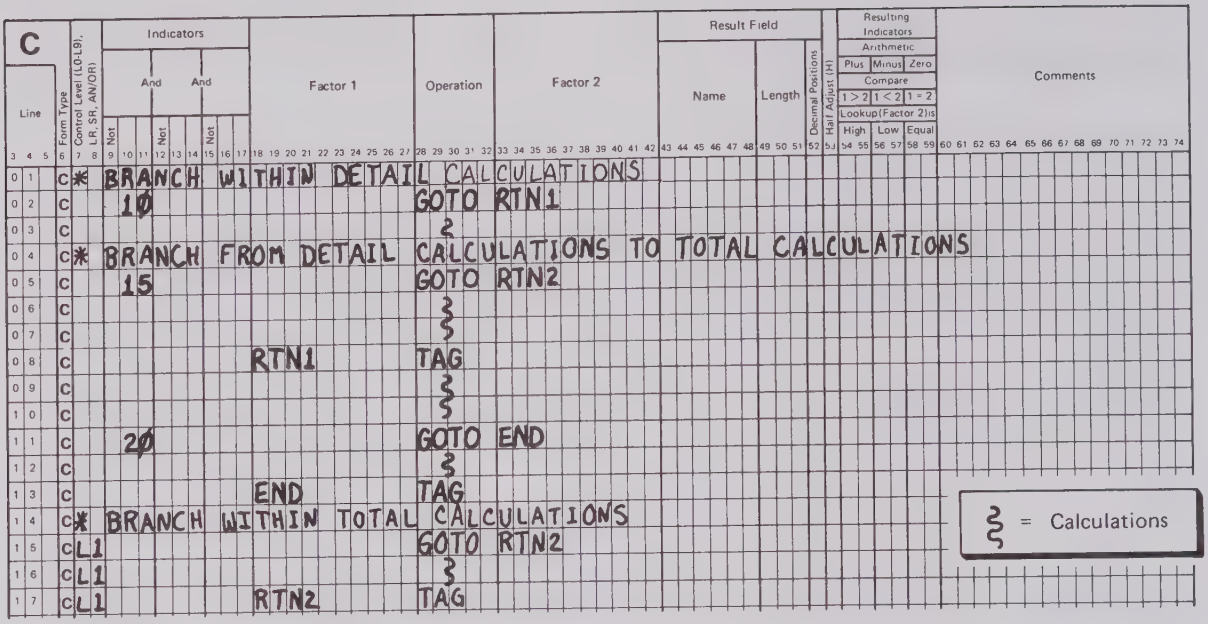
A branch cannot be made from a total calculation line to a detail calculation line. (A total calculation line is defined as one that is conditioned by a control level indicator in positions 7 and 8 of the calculation specifications.)

Branching from one part of the RPG/400 logic cycle to another may result in an endless loop. You are responsible for ensuring that the logic of your program does not produce undesirable results.

Factor 2 must contain the label to which the program is to branch. This label is entered in factor 1 of a TAG or ENDSR operation. The label must be a unique symbolic name.

See Figure 9-19 on page 9-74 for examples of the GOTO operation.

GOTO (Branch To)



If indicator 10, 15, or 20 is on, the program branches to the TAG label specified in the GOTO operations.

Figure 9-19. GOTO and TAG Operations

IFxx (If/Then)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Blank	Blank	Blank	Blank

The IFxx operation allows a group of calculations to be processed if a certain relationship, specified by xx, exists between factor 1 and factor 2. When ANDxx and ORxx operations are used with the IFxx, the group of calculations is performed if the condition specified by the combined operations exists. (For the meaning of xx, see "Structured Programming Operations" on page 9-11.)

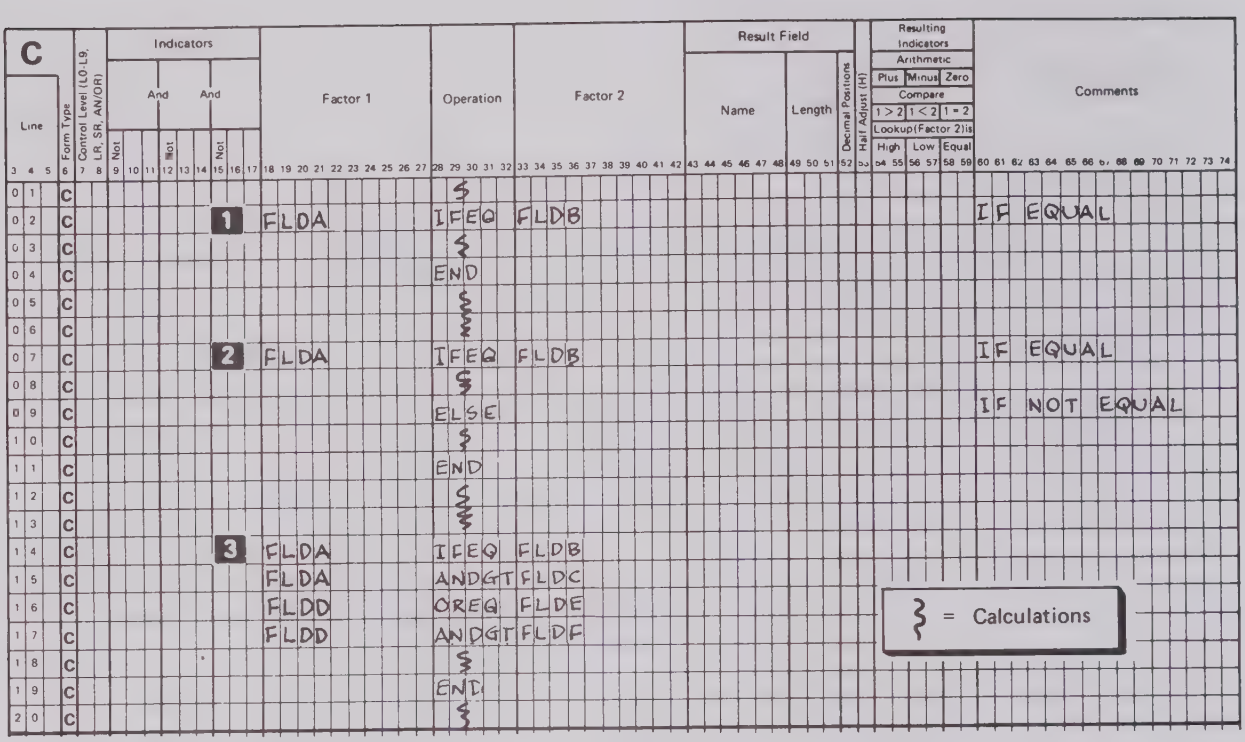
Conditioning indicators can be used. Factor 1 and factor 2 must contain a character literal, a numeric literal, or a field name. Both the factor 1 and factor 2 entries must be character data, or both must be numeric.

If the relationship between factor 1 and factor 2 or the condition specified by the combined IFxx, ANDxx, or ORxx operations does not exist, control passes to the calculation operation immediately following the associated END operation. However, if the relationship between factor 1 and factor 2 or the condition specified by IFxx, ANDxx, or ORxx operations does not exist and an ELSE operation is specified, control passes to the first calculation operation that can be processed following the ELSE operation.

Conditioning indicator entries on the END operation associated with IFxx must be blank.

An END statement must be used to close an IFxx group. If an IFxx statement is followed by an ELSE statement, an END statement is needed after the ELSE statement but not after the IFxx statement.

Figure 9-20 on page 9-76 illustrates uses of the IFxx operation.



1 If FLDA equals FLDB (line 02), the calculation in line 03 is processed. If FLDA does not equal FLDB, the program branches to the operation immediately following the END statement (line 05).

2 If FLDA equals FLDB (line 07), the calculation in line 08 is processed and control passes to the operation immediately following the END statement (line 12). If FLDA does not equal FLDB, control passes to the ELSE statement (line 09) and the calculation in line 10 is processed.

3 If either of the specified conditions:

- FLDA equal to FLDB and greater than FLDC, or
- FLDD equal to FLDE and greater than FLDF

is met, the calculation in line 18 is processed. If neither of the specified conditions exists, the program branches to the operation immediately following the END statement (line 20).

Figure 9-20. IFxx/END and IFxx/ELSE/END Operations

IN (Retrieve a Data Area)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Required	Blank	Blank	Opt: ER	Blank

The IN operation retrieves a data area and optionally allows you to specify whether the data area is to be locked from update by another program. In order for a data area to be retrieved by the IN operation, it must be specified in the result field of an *NAMVAR DEFN statement (see "DEFN (Field Definition)" on page 9-42 for information on *NAMVAR DEFN).

Factor 1 can contain the reserved word *LOCK or can be blank. *LOCK indicates that the data area cannot be updated or locked by another program until (1) an UNLCK operation is processed, (2) an OUT operation with no factor 1 entry is processed, or (3) RPG III implicitly unlocks the data area at end of program.

Factor 1 must be blank when factor 2 contains the name of the local data area.

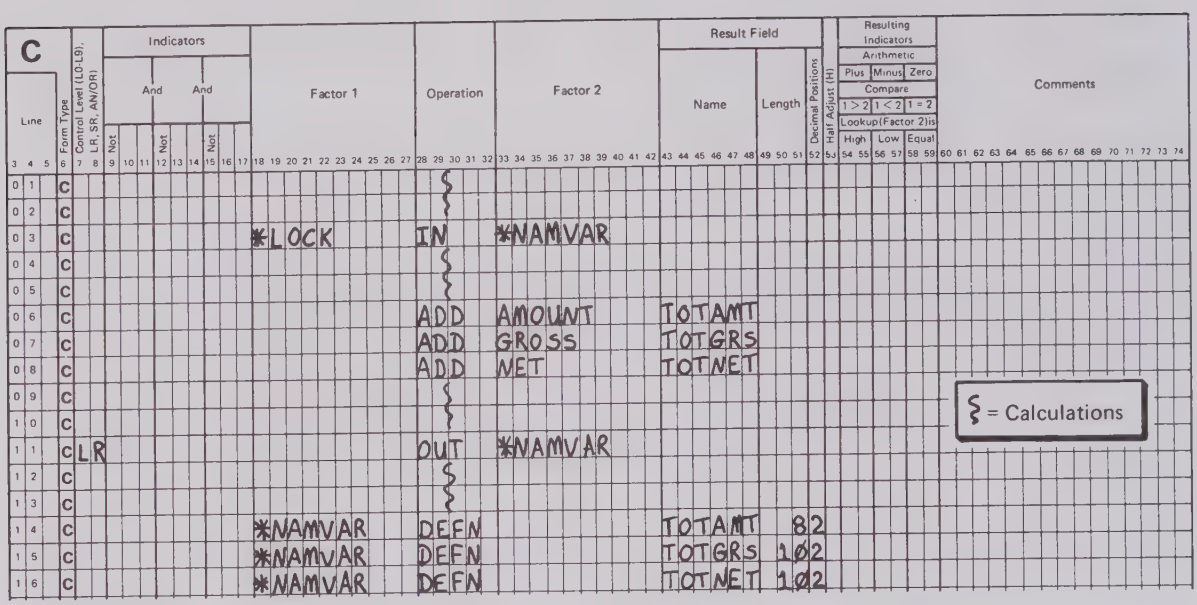
An *LOCK IN statement can be specified for a data area that is already locked. When factor 1 is blank, the lock status is the same as it was before the data area was retrieved; if it was locked, it remains locked; if it was unlocked, it remains unlocked.

Factor 2 must be either the name of the data area to be retrieved or the reserved word *NAMVAR. When *NAMVAR is specified, all data areas defined in the program are retrieved. If an error occurs on the retrieval of a data area (for example, a data area can be retrieved but cannot be locked), an error occurs on the IN operation and the RPG/400 exception/error handling routine receives control. If a program status data structure (*PSSR) is specified, the data structure contains information on the data area in error. If a message is issued to the requester, the message identifies the data area in error.

A resulting indicator can be specified in positions 56 and 57 to be set on if an error occurs during the operation. Positions 54-55 and 58-59 must be blank.

For further rules for the IN operation, see "Data Area Operations" on page 9-15. See Figure 9-21 on page 9-78 for an example of the IN operation.

IN (Retrieve a Data Area)



TOTAMT, TOTGRS, and TOTNET, which are specified in lines 14 through 16, are defined as data areas in the system. The IN operation (line 3) retrieves all the data areas defined in the program and locks them. The program processes calculations (lines 4 through 10), and at LR time it writes and unlocks all the data areas (line 11). The data area can then be used by other programs.

Figure 9-21. IN and OUT Operations

KFLD (Define Parts of a Key)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Blank	Blank	Blank	Required	Blank	Blank	Blank

The KFLD operation is a declarative (nonprocessed) operation that indicates that a field is part of a search argument identified by a KLIST name.

The KFLD operation can be specified anywhere within calculations, including total calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. Conditioning indicator entries (positions 9 through 17) are not permitted.

The result field must contain the name of a field that is to be part of the search argument. The result field cannot contain an array name or a table name. Each KFLD field must agree in length, data type (character or numeric), and decimal position with the corresponding field in the composite key of the record or file. However, each KFLD field need not have the same name as the corresponding field in the composite key. The order in which the KFLD fields are specified in the KLIST determines which KFLD is associated with a particular field in the composite key. For example, the first KFLD field following a KLIST operation is associated with the leftmost (high-order) field of the composite key.

Figure 9-22 on page 9-81 shows an example of the KLIST operation with KFLD operations.

KLIST (Define A Composite Key)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Blank	Required	Blank	Blank	Blank	Blank	Blank

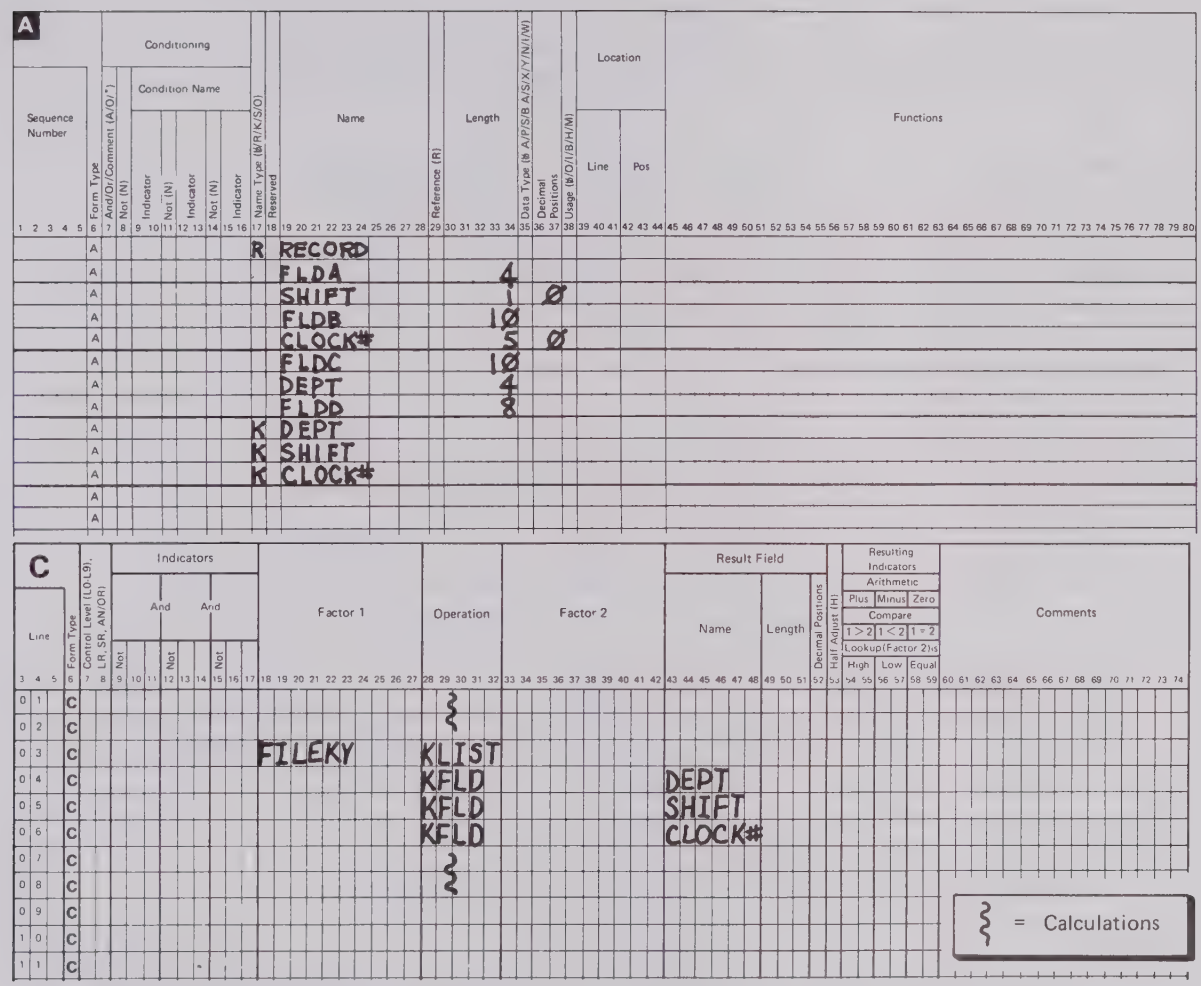
The KLIST operation is a declarative operation that gives a name to a list of KFLDs. This list can be used as a search argument to retrieve records from files that have a composite key.

A KLIST operation can be specified anywhere within calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. Conditioning indicator entries (positions 9 through 17) are not permitted. Factor 1 must contain a KLIST name that must be a unique symbolic name.

Remember the following when specifying a KLIST operation:

- If a search argument is composed of more than one field (a composite key), you must specify a KLIST with multiple KFLDs.
- A KLIST name can be specified as a search argument only for externally described files.
- A KLIST and its associated KFLD fields can appear anywhere in calculations.
- A KLIST must be followed immediately by at least one KFLD.
- A KLIST is ended when a non-KFLD operation is encountered.
- A KLIST name can appear in factor 1 of a CHAIN, DELET, READE, SETGT, or SETLL operation.
- The same KLIST name can be used as the search argument for multiple files, or it can be used multiple times as the search argument for the same file.

Figure 9-22 on page 9-81 shows an example of the KLIST operation.



The KLIST operation (line 03) indicates the name, FILEKY, by which the search argument (lines 04, 05, and 06) can be specified. The following diagram shows what the search argument looks like. The fields DEPT, SHIFT, and CLOCK# are key fields in this record.

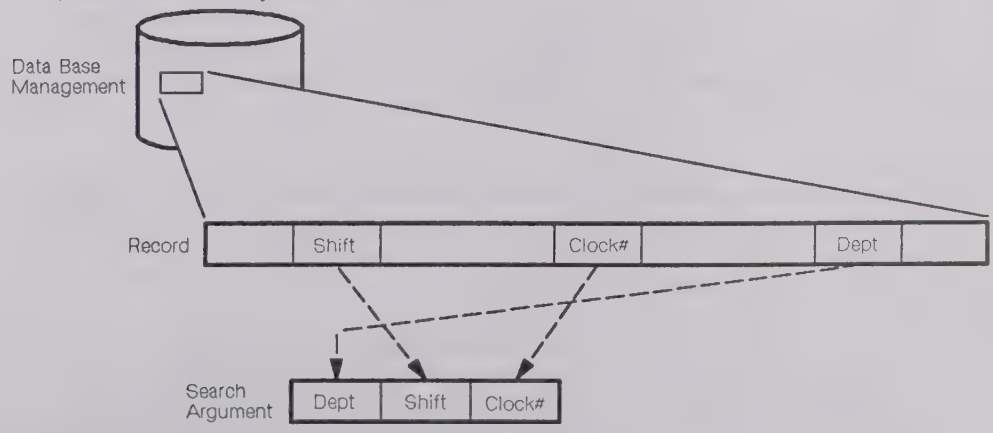


Figure 9-22. KLIST and KFLD Operations

LOKUP (Lookup)

Array LOKUP

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Blank	Opt: HI	Opt: LO	Opt: EQ

Table LOKUP

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Optional	Opt: HI	Opt: LO	Opt: EQ

The LOKUP operation causes a search to be made for a particular element in an array or table. Factor 1 is the search argument (data for which you want to find a match in the array or table named). Factor 1, the search argument, can be:

- A character or numeric literal
- A field name
- An array element
- A table name.

If a table is named in factor 1, the search argument used is the element of the table last selected in a LOKUP operation, or it is the first element of the table if a previous LOKUP has not been processed. The array or table to be searched is specified in factor 2.

For a table LOKUP, the result field can contain the name of a second table from which an element (corresponding positionally with that of the first table) can be retrieved. The name of the second table can be used to reference the element retrieved. The result field must be blank if factor 2 contains an array name.

Resulting indicators specify the search condition for the LOKUP. One resulting indicator must be specified in positions 54 through 59. A second resulting indicator is allowed if one is specified in positions 58 and 59 (equal).

Decimal alignment is not processed for LOKUP operations.

At least one resulting indicator must be specified in positions 54 through 59 first to determine the search to be done and then to reflect the result of the search. The specified indicator(s) is set on only if the search is successful. No more than two indicators can be used. Resulting indicators can be assigned to equal and high or to

equal and low. The program searches for an entry that satisfies either condition with equal given precedence; that is, if no equal entry is found, the nearest lower or nearest higher entry is selected.

Resulting indicators specify the type of search and reflect the result of the search in the following manner:

High: Instructs the program to find the entry that is nearest to, yet higher in sequence than, the search argument. The first higher entry found sets the indicator assigned to *high* on.

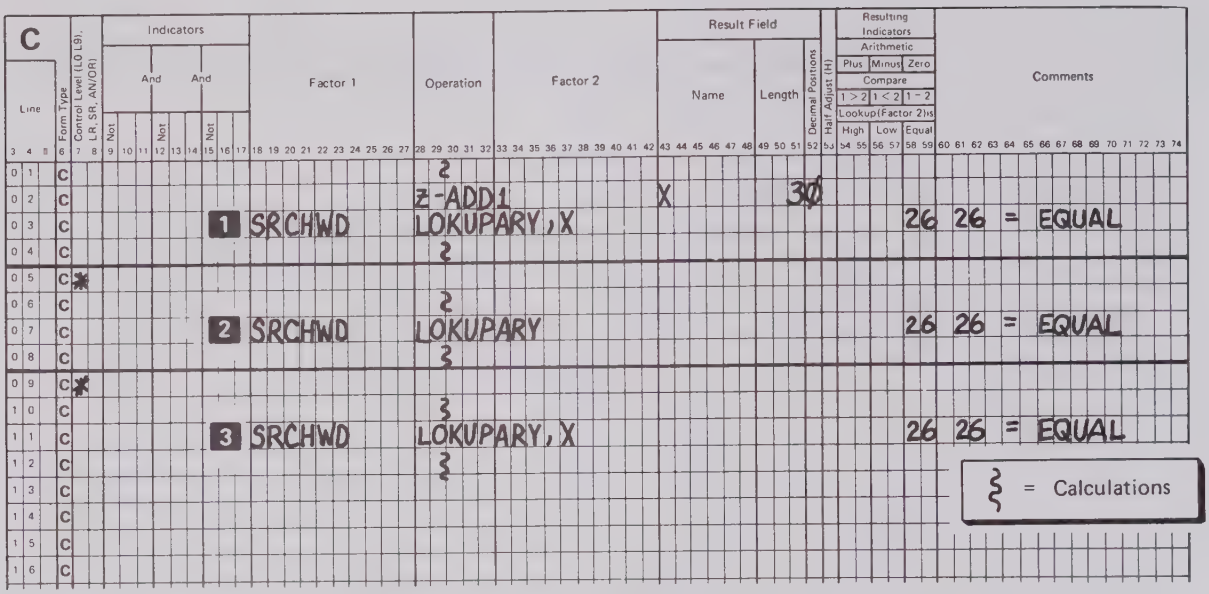
Low: Instructs the program to find the entry that is nearest to, yet lower in sequence than, the search argument. The first such entry found sets the indicator assigned to *low* on.

Equal: Instructs the program to find the entry equal to the search argument. The first equal entry found sets the indicator assigned to *equal* on.

When you use the LOKUP operation, remember:

- The search argument and array element or table element must have the same length and the same format (character or numeric).
- When LOKUP is processed on an array and an index is used, the LOKUP begins with the element specified by the index. The index value is set to the position number of the element located. An error occurs if the index is equal to zero or is higher than the number of elements in the array when the search begins. The index is set equal to one if the search is unsuccessful.
- A search can be made for high, low, high and equal, or low and equal only if a sequence is specified for the array or table in the extension specifications. For further information on the sequence entry, see "Position 45 (Sequence)" on page 3-9.
- No resulting indicator is set on if the search is not successful.

Figure 9-23 on page 9-84 shows examples of the LOKUP operation with arrays.



1 In this example, the programmer wants to know which element in ARY the LOKUP operation locates. The Z-ADD operation sets the field X to 1. The LOKUP starts at the element of ARY that is indicated by field X and continues running until it finds the first element equal to SRCHWD. The index value, X, is set to the position number of the element located.

2 In this example, the programmer wants to know when an element is found that is equal to SRCHWD. LOKUP searches ARY until it finds the first element equal to SRCHWD. When this occurs, indicator 26 is set on.

3 This example is identical to **1**. The LOKUP starts at a variable index number specified by field X. Field X does not have to be set to 1 before the LOKUP operation. When LOKUP locates the first element in ARY equal to SRCHWD, indicator 26 is set on. The index value, X, is set to the position number of the element located.

Figure 9-23. LOKUP Operation with Arrays

MHHZO (Move High To High Zone)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Blank	Blank	Blank

The MHHZO operation moves the zone from the leftmost position of factor 2 to the leftmost position of the result field. Factor 2 and the result field must be defined as character fields. For further information on the MHHZO operation, see "Move Zone Operations" on page 9-9.

The function of the MHHZO operation is shown in Figure 9-24 on page 9-88.

MHLZO (Move High to Low Zone)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Blank	Blank	Blank

The MHLZO operation moves the zone from the leftmost position of factor 2 to the rightmost position of the result field. Factor 2 must be defined as a character field. The result field can be character or numeric data. For further information on the MHLZO operation, see "Move Zone Operations" on page 9-9.

The function of the MHLZO operation is shown in Figure 9-24 on page 9-88.

MLHZO (Move Low to High Zone)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Blank	Blank	Blank

The MLHZO operation moves the zone from the rightmost position of factor 2 to the leftmost position of the result field. Factor 2 can be defined as a numeric field or as a character field, but the result field must be a character field. For further information on the MLHZO operation, see "Move Zone Operations" on page 9-9.

The function of the MLHZO operation is shown in Figure 9-24 on page 9-88.

MLLZO (Move Low to Low Zone)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Blank	Blank	Blank

The MLLZO operation moves the zone from the rightmost position of factor 2 to the rightmost position of the result field. Factor 2 and the result field can be either character data or numeric data. For further information on the MLLZO, see "Move Zone Operations" on page 9-9.

The function of the MLLZO operation is shown in Figure 9-24.

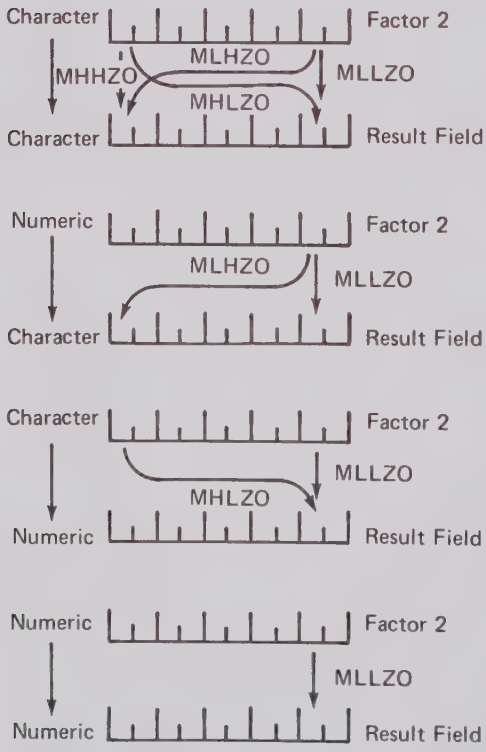


Figure 9-24. Function of MOVE Zone Operations

MOVE (Move)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Opt: +	Opt: –	Opt: ZB

The MOVE operation transfers characters from factor 2 to the result field. Moving starts with the rightmost character of factor 2.

If factor 2 is longer than the result field, the excess leftmost characters of factor 2 are not moved. If the result field is longer than factor 2, the excess leftmost characters in the result field are unchanged.

You cannot specify resulting indicators if the result field is an array. However, you can specify them if the result field is an array element, or non-array field.

For further information on the MOVE operation, see "Move Operations" on page 9-8.

The MOVE operation is summarized in Figure 9-25 on page 9-90.

Factor 2 Shorter Than Result Field

	Factor 2		Result Field	
a. Character	P H 4 S N	Before MOVE	1 2 3 4 5 6 7 8 +4	Character
	P H 4 S N	After MOVE	1 2 3 4 P H 4 S N	
b. Character	P H 4 S N	Before MOVE	1 2 3 4 5 6 7 8 +4	Numeric
	P H 4 S N	After MOVE	1 2 3 4 7 8 4 2 −5	
c. Numeric	1 2 7 8 4 2 5	Before MOVE	1 2 3 4 5 6 7 8 9	Numeric
	1 2 7 8 4 2 5	After MOVE	1 2 1 2 7 8 4 2 5	
d. Numeric	1 2 7 8 4 2 5	Before Move	A C F G P H 4 S N	Character
	1 2 7 8 4 2 5	After MOVE	A C 1 2 7 8 4 2 5	

Factor 2 Longer Than Result Field

	Factor 2		Result Field	
a. Character	A C E G P H 4 S N	Before MOVE	5 6 7 8 4	Character
	A C E G P H 4 S N	After MOVE	P H 4 S N	
b. Character	A C E G P H 4 S N	Before MOVE	5 6 7 8 +4	Numeric
	A C E G P H 4 S N	After MOVE	7 8 4 2 −5	
c. Numeric	1 2 7 8 4 2 5	Before MOVE	5 6 7 4 8	Numeric
	1 2 7 8 4 2 5	After MOVE	7 8 4 2 5	
d. Numeric	1 2 7 8 4 2 5	Before MOVE	P H 4 S N	Character
	1 2 7 8 4 2 5	After MOVE	7 8 4 2 5	

Factor 2 and Result Field Same Length

	Factor 2		Result Field	
a. Character	P H 4 S N	Before MOVE	5 6 7 8 4	Character
	P H 4 S N	After MOVE	P H 4 S N	
b. Character	P H 4 S N	Before MOVE	5 6 7 8 4	Numeric
	P H 4 S N	After MOVE	7 8 4 2 −5	
c. Numeric	7 8 4 2 −5	Before MOVE	A L T 5 F	Numeric
	7 8 4 2 −5	After MOVE	7 8 4 2 −5	
d. Numeric	7 8 4 2 −5	Before MOVE	A L T 5 F	Character
	7 8 4 2 −5	After MOVE	7 8 4 2 N	

Note: +4 = letter D , and −5 = letter N.

Figure 9-25. MOVE Operations

MOVEA (Move Array)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Opt: +	Opt: –	Opt: ZB

The MOVEA operation transfers character or numeric values from factor 2 to the result field. (Certain restrictions apply when moving numeric values.) Factor 2 or the result field must contain an array. Factor 2 and the result field cannot specify the same array even if the array is indexed. Resulting indicators can be specified to test the result field except when the result field is an array or array element.

You can use MOVEA with a packed, binary, zoned, or character array.

The MOVEA operation makes it possible to:

- Move several contiguous character array elements to a single character field
- Move a single character field to several contiguous character array elements.
- Move contiguous array elements to contiguous elements of another array.

Movement of data starts with the first element of an array if the array is not indexed or with the element specified if the array is indexed. The movement of data is ended when the last array element is moved or filled.

The coding for and results of MOVEA operations are shown in Figure 9-26 on page 9-93.

Character MOVEA Operations:

Both factor 2 and the result field must be defined as character.

On a character MOVEA operation, movement of data ends when the number of characters moved equals the shorter length of the fields specified by factor 2 and the result field; therefore, the character MOVEA operation could end in the middle of an array element.

Numeric MOVEA Operations:

Moves are only valid between fields and array elements with the same numeric length defined. Factor 2 and the result field entries may specify numeric fields, numeric array elements, or numeric arrays, but at least one must be an array or array element. The numeric types may be binary, packed decimal, or zoned decimal but need not be the same between factor 2 and the result field.

Factor 2 can contain a numeric literal if the result field entry specifies a numeric array or numeric array-element:

- The numeric literal may not contain a decimal point.

- The length of the numeric literal cannot be greater than the element length of the array or array element specified in the result field.

Decimal positions are ignored during the move and need not correspond. Numeric values are not converted to account for the differences in the defined number of decimal places.

The figurative constant *ALL'x' is not valid on numeric MOVEA operations.

General MOVEA Operations

Applications which require a MOVEA operation within an array of numeric fields but which cannot be performed due to the restrictions on numeric MOVEA operations might possibly be performed through character MOVEA operations by performing the following:

- Define the numeric array as a subfield of a data structure
- Redefine the numeric array in the data structure as a character array.

The numeric array must be in zoned decimal format.

If a figurative constant is specified with MOVEA, the length of the constant generated is equal to the portion of the array specified. Examples are:

- MOVEA *BLANK ARR,X

  Beginning with element X, the remainder of ARR will contain blanks.

- MOVEA *ALL'XYZ' ARR,X

  ARR has 4-byte character elements. Element boundaries are ignored, as is always the case with MOVEA. Beginning with element X, the remainder of the array will contain 'XYZXYZXYZXYZ. . .'.

Note that the results of MOVEA are different from those of the MOVE example above.

For further information on the MOVEA operation, see "Move Operations" on page 9-8.

Example: Array-to-array move. No indexing; different length arrays, same element length.

Calculation Specifications

	Operation	Factor 2	Result Field Name	Result Field Length	Result Field Decimal Positions
	MOVEA	ARRX	ARRY		

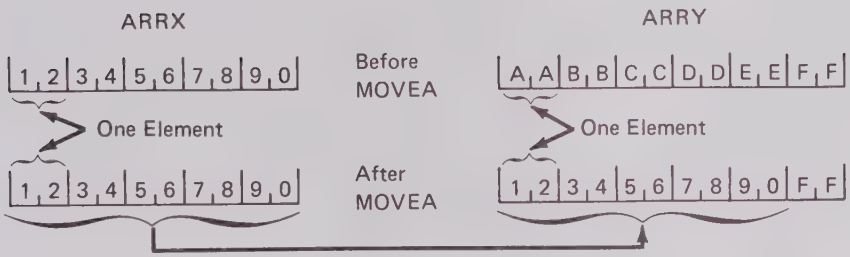
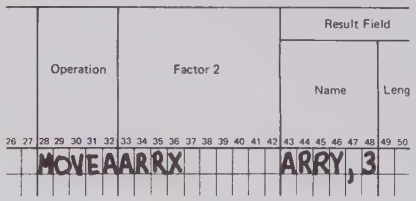
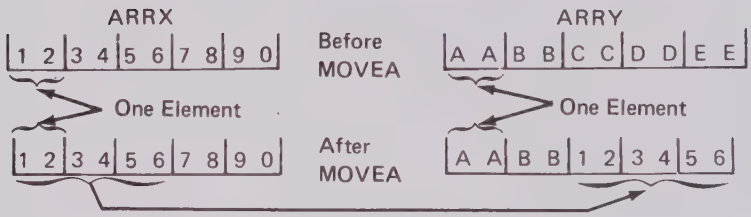


Figure 9-26 (Part 1 of 3). MOVEA Operations

Example: Array-to-array move. Index result field.

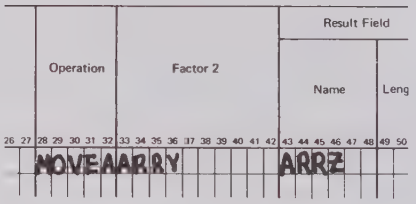
Calculation Specifications

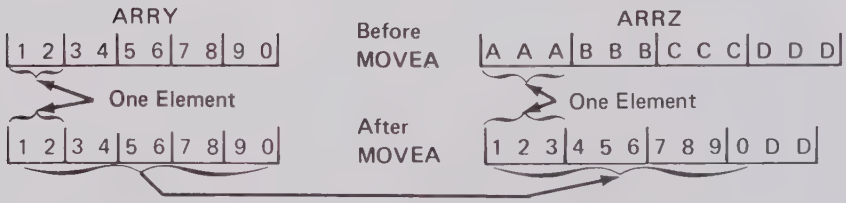




Example: Array-to-array move. No indexing, different length array elements.

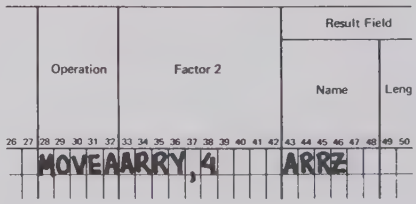
Calculation Specifications





Example: Array-to-array move. Index factor 2, different length array elements.

Calculation Specifications



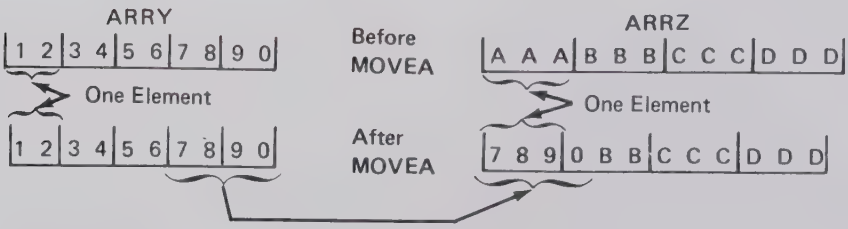
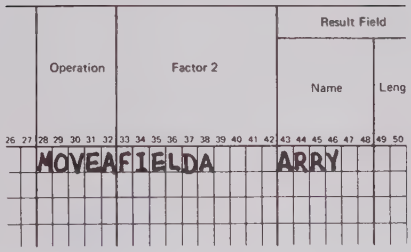
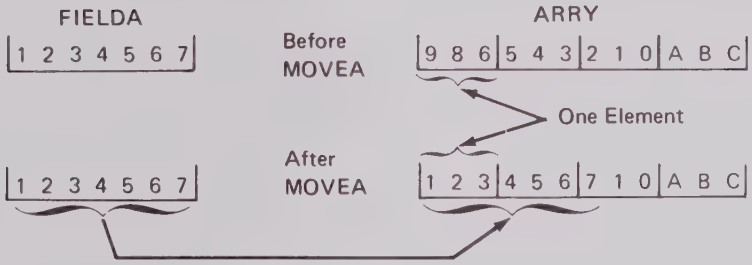


Figure 9-26 (Part 2 of 3). MOVEA Operations

Example: Field-to-array move.
No indexing on array.

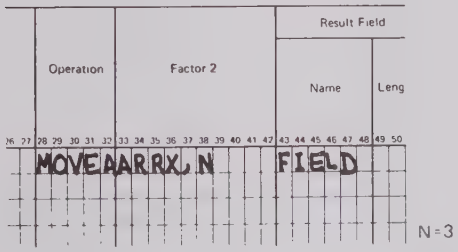
Calculation Specifications





Example: Array-to-field move.
Variable indexing.

Calculation Specifications



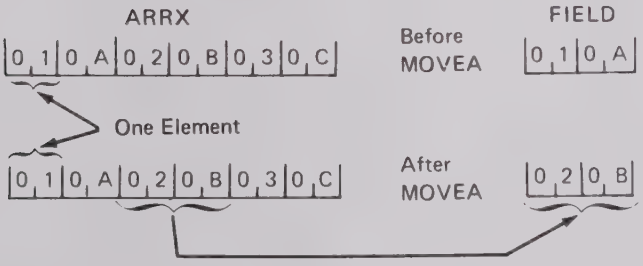


Figure 9-26 (Part 3 of 3). MOVEA Operations

MOVEL (Move Left)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Opt: +	Opt: –	Opt: ZB

The MOVEL operation transfers characters from factor 2 to the result field. Moving begins with the leftmost character in factor 2. You cannot specify resulting indicators if the result field is an array. However, you can specify them if the result field is an array element, or non-array field.

When data is moved to a numeric field, the sign (+ or -) of the result field is retained except when factor 2 is as long as or longer than the result field. In this case, the sign of factor 2 is used as the sign of the result field.

The MOVEL operation is summarized in Figure 9-27 on page 9-97.

A summary of the rules for MOVEL operation for three conditions based on field lengths is as follows:

1. Factor 2 is the same length as the result field:
   a. If factor 2 and the result field are numeric, the sign is moved with the rightmost position.
   b. If factor 2 is numeric and the result field is character, the sign is moved with the rightmost position.
   c. If factor 2 is character and the result field is numeric, a minus zone is moved into the rightmost position of the result field if the zone from the rightmost position of factor 2 is a hexadecimal D (minus zone). However, if the zone from the rightmost position of factor 2 is not a hexadecimal D, a positive zone is moved into the rightmost position of the result field. Digit portions are converted to their corresponding numeric characters. If the digit portions are not valid digits, a data exception error occurs.
   d. If factor 2 and the result field are character, all characters are moved.
2. Factor 2 is longer than the result field:
   a. If factor 2 and the result field are numeric, the sign from the rightmost position of factor 2 is moved into the rightmost position of the result field.
   b. If factor 2 is numeric and the result field is character, the result field contains only numeric characters.
   c. If factor 2 is character and the result field is numeric, a minus zone is moved into the rightmost position of the result field if the zone from the rightmost position of factor 2 is a hexadecimal D (minus zone). However, if the zone from the rightmost position of factor 2 is not a hexadecimal D, a positive zone is moved into the rightmost position of the result field. Other result field positions contain only numeric characters.

d. If factor 2 and the result field are character, only the number of characters needed to fill the result field are moved.

Factor 2 and Result Field Same Length

	Factor 2		Result Field	
a. Numeric	7 8↓4 2 5	Before MOVEL	5 6 7↓8 +4	Numeric
	7 8↓4 2 5	After MOVEL	7 8 4↓2 5	
b. Numeric	7 8↓4 2 5	Before MOVEL	A K T 4 D	Character
	7 8↓4 2 5	After MOVEL	7 8 4 2 N	
c. Character	P H 4 S N	Before MOVEL	5 6 7 8 +4	Numeric
	P H 4 S N	After MOVEL	7 8 4 2 5	
d. Character	P H 4 S N	Before MOVEL	A K T 4 D	Character
	P H 4 S N	After MOVEL	P H 4 S N	

Factor 2 Longer Than Result Field

	Factor 2		Result Field	
a. Numeric	0 0 0 2 5 8 4 2 5	Before MOVEL	5↓6 7 8 +4	Numeric
	0 0 0 2 5 8 4 2 5	After MOVEL	0↓0 0 2 5	
b. Numeric	9 0 3 1 7 8 4 2 5	Before MOVEL	A K T 4 D	Character
	9 0 3 1 7 8 4 2 5	After MOVEL	9 0 3 1 7	
c. Character	B R W C X H 4 S N	Before MOVEL	5 6 7 8↓+4	Numeric
	B R W C X H 4 S N	After MOVEL	2 9 6 3↓7	
d. Character	B R W C X H 4 S N	Before MOVEL	A K T 4 D	Character
	B R W C X H 4 S N	After MOVEL	B R W C X	

Factor 2 Shorter Than Result Field

		Factor 2		Result Field	
a.	Numeric	7 8 4 2 5	Before MOVEL	1↓3 0 9 4 3 2 1 +0	Numeric
		7 8 4 2 5	After MOVEL	7↓8 4 2 5 3 2 1 +0	
	Character	C P T 5 N	Before MOVEL	1 3 0 9 4 3 2 1 +0	Numeric
		C P T 5 N	After MOVEL	3 7 3 5 5 3 2 1 +0	
b.	Numeric	7 8 4 2 5	Before MOVEL	B R W C X H 4 S A	Character
		7 8 4 2 5	After MOVEL	7 8 4 2 N H 4 S A	
	Character	C P T 5 N	Before MOVEL	B R W C X H 4 S A	Character
		C P T 5 N	After MOVEL	C P T 5 N H 4 S A	

Note: $\overset{+}{4}$ = letter D, and $\overset{-}{5}$ = letter N; arrow ↓ is decimal point.

Figure 9-27. MOVEL Operations

3. Factor 2 is shorter than the result field:

a. If factor 2 is either numeric or character and the result field is numeric, the digit portion of factor 2 replaces the contents of the leftmost positions of the result field. The sign in the rightmost position of the result field is not changed.
b. If factor 2 is either numeric or character and the result field is character data, the characters in factor 2 replace the equivalent number of leftmost positions in the result field. No change is made in the zone of the rightmost position of the result field.

For further information on the MOVEL operation, see "Move Operations" on page 9-8.

MULT (Multiply)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Required	Required	Opt: +	Opt: –	Opt: Z

If factor 1 is specified, factor 1 is multiplied by factor 2 and the product is placed in the result field. Be sure that the result field is large enough to hold the product. Use the following rule to determine the maximum result field length: result field length equals the length of factor 1 plus the length of factor 2. If factor 1 is not specified, factor 2 is multiplied by the result field and the product is placed in the result field.

For further information on the MULT operation, see "Arithmetic Operations" on page 9-5.

See Figure 9-1 on page 9-7 for examples of the MULT operation.

MVR (Move Remainder)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Blank	Required	Opt: +	Opt: –	Opt: Z

The MVR operation moves the remainder from the previous DIV operation to a separate field named as the result field. Factor 1 and factor 2 must be blank. The MVR operation must immediately follow the DIV operation. If you use conditioning indicators, it is your responsibility to ensure that the MVR operation is processed immediately after the DIV operation. If the MVR operation is processed before the DIV operation, undesirable results occur. For example, error message RPG0907 may be issued.

Remember to leave sufficient room in the result field if the DIV operation uses factors with decimal positions. The number of significant decimal positions is the greater of:

- The number of decimal positions in factor 1 of the previous divide operation
- The sum of the decimal positions in factor 2 and the result field of the previous divide operation.

The sign (+ or -) of the remainder is the same as the dividend (factor 1).

Half adjust (positions 53) cannot be specified on a DIV operation that is immediately followed by an MVR operation.

The MVR operation cannot be used if the previous divide operation has an array specified in the result field.

For further information on the MVR operation, see "Arithmetic Operations" on page 9-5.

See Figure 9-1 on page 9-7 for an example of the MVR operation.

NEXT (Next)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Blank	Blank	Opt: ER	Blank

The NEXT operation code forces the next input for a multiple device file to come from the program device specified in factor 1, providing the input operation is a cycle read or a READ-by-file-name. Any read operation, including CHAIN, EXFMT, READ, and READC, ends the effect of the previous NEXT operation. If NEXT is specified more than once between input operations, only the last operation is processed. The NEXT operation code can be used only for a multiple device file.

To use this operation, enter NEXT in positions 28 through 32. In factor 1, enter the name of a 10-character field that contains the program device name or an alphameric literal that is the program device name. In factor 2, enter the name of the multiple device WORKSTN file for which the operation is requested.

An indicator can be specified in positions 56 and 57. This indicator is set on if an exception/error occurs on the NEXT operation. If the INFSR subroutine is specified and positions 56 and 57 do not contain an indicator, the subroutine automatically receives control when an exception/error occurs. If the INFSR subroutine is not specified and positions 56 and 57 do not contain an indicator, the default error handler takes control.

OCUR (Set/Get Occurrence of a Data Structure)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Required	Optional	Blank	Opt: ER	Blank

The OCUR operation code specifies which occurrence of the data structure is to be used next within an RPG/400 program. If a data structure with multiple occurrences or a subfield of that data structure is specified in an operation, the first occurrence of the data structure is used until an OCUR operation is specified. After an OCUR operation is specified, the occurrence of the data structure that was established by the OCUR operation is used.

Factor 1 is optional; if specified, it may contain a numeric literal or a numeric field name with zero decimal positions, or a data structure name. Factor 1 is used during the OCUR operation to set the occurrence of the data structure specified in factor 2. If factor 1 is blank, the value of the current occurrence of the data structure in factor 2 is placed in the result field during the OCUR operation.

If factor 1 is a data structure name, it must be a multiple occurrence data structure. The current occurrence of the data structure in factor 1 is used to set the occurrence of the data structure in factor 2.

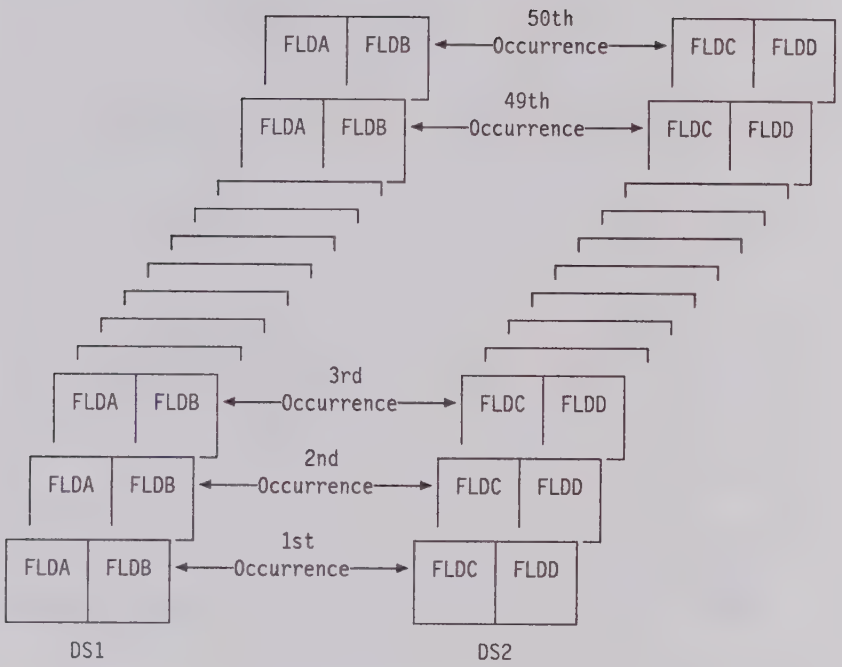
Factor 2 is required and must be the name of a multiple occurrence data structure.

The result field is optional; if specified, it must be a numeric field name with zero decimal positions. During the OCUR operation, the value of the current occurrence of the data structure specified in factor 2, after being set by any value or data structure that is optionally specified in factor 1, is placed in the result field.

A resulting indicator can be specified in positions 56 and 57 to be set on if the occurrence specified is outside the valid range set for the data structure. If the occurrence is outside the valid range, the occurrence of the data structure in factor 2 remains the same as before the OCUR operation processed.

Figure 9-28 on page 9-103 illustrates uses of the OCUR operation.

The OCUR operation establishes which occurrence of a multiple occurrence data structure is used next in a program. Only one occurrence can be used at a time.



DS1 and DS2 (in the preceding diagram specified in lines 01 and 05 of the input specifications) are multiple occurrence data structures. Each data structure has 50 occurrences.

I

Line	Form Type	Filename or Record Name	Sequence	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, **, or DS	External Field Name / Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location: From	Field Location: To	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Field Indicators: Minus	Field Indicators: Zero or Blank
		Data Structure Name	O A N	R D																	Data Structure: Occurs n Times	Data Structure: Length								
3 4 5	6	7 8 9 10 11 12	13 14 15	16 17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70
01	I	DS1				DS															50									
02	I																				1	5		FLDA						
03	I																				6	8Ø		FLDB						
04	I																													
05	I	DS2				DS															50									
06	I																				1	6		FLDC						
07	I																				7	11		FLDD						
08	I																													

Figure 9-28 (Part 1 of 2). Uses of the OCUR Operation

OCUR (Set/Get Occurrence of a Data Structure)

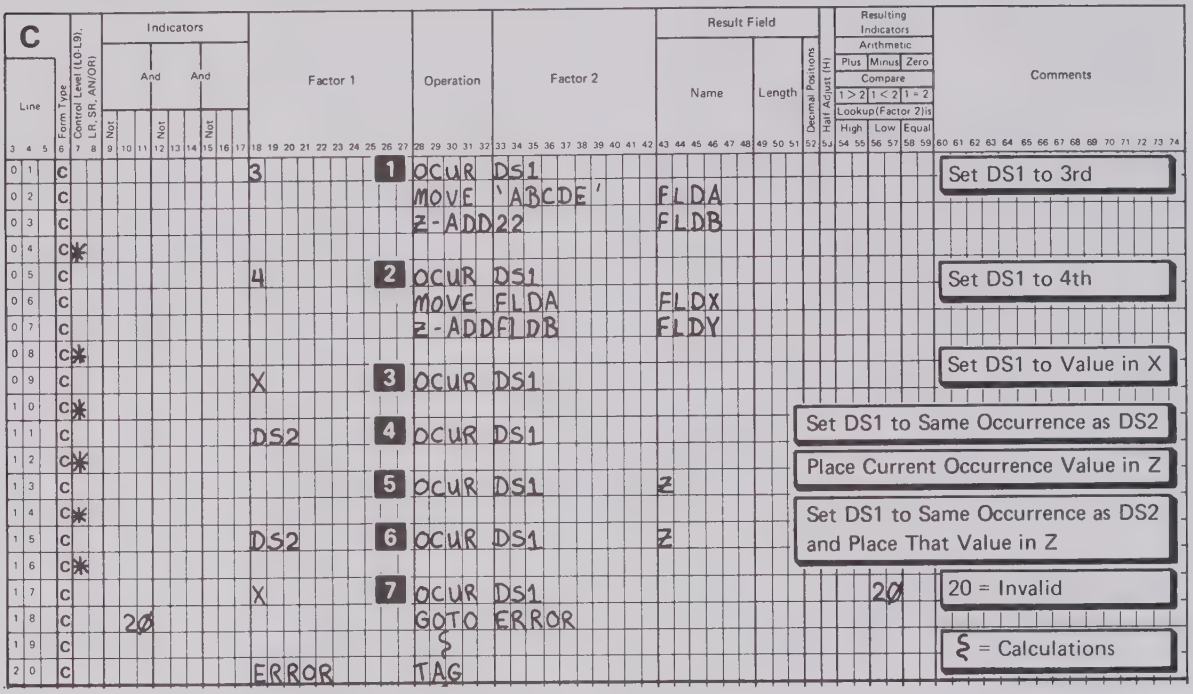


Figure 9-28 (Part 2 of 2). Uses of the OCUR Operation

1 DS 1 is set to the third occurrence. The subfields FLDA and FLDB of the third occurrence can now be used. The MOVE and Z-ADD operations change the contents of FLDA and FLDB, respectively, in the third occurrence of DS1.

2 DS1 is set to the fourth occurrence. Using the values in FLDA and FLDB of the fourth occurrence of DS1, the MOVE operation places the contents of FLDA in the result field, FLDX, and the Z-ADD operation places the contents of FLDB in the result field, FLDY.

3 DS1 is set to the occurrence specified in field X. For example, if X = 10, DS1 is set to the tenth occurrence.

4 DS1 is set to the current occurrence of DS2. For example, if the current occurrence of DS2 is the twelfth occurrence, DS1 is set to the twelfth occurrence.

5 The value of the current occurrence of DS1 is placed in the result field, Z. Field Z must be numeric with zero decimal positions. For example, if the current occurrence of DS1 is 15, field Z contains the value 15.

6 DS1 is set to the current occurrence of DS2. The value of the current occurrence of DS1 is then moved to the result field, Z. For example, if the current occurrence of DS2 is the fifth occurrence, DS1 is set to the fifth occurrence. The result field, Z, contains the value 5.

7 DS1 is set to the current occurrence of X. For example, if X = 15, DS1 is set to the fifteenth occurrence. If X is equal to 0 or greater than 50, an error occurs and indicator 20 is set on. The program then branches to the TAG specified in the GOTO operation.

OPEN (Open File for Processing)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Opt: ER	Blank

The explicit OPEN operation opens the file named in factor 2. The factor 2 entry cannot be designated as a primary, secondary, or table file. A resulting indicator can be specified in positions 56 and 57 to be set on if the OPEN operation is not successful. If no indicator is specified, but the INFSR subroutine is specified, the INFSR automatically receives control when an error/exception occurs. If no indicator or INFSR subroutine is specified, the default error/exception handler receives control when an error/exception occurs.

To open the file specified in factor 2 for the first time in a program with an explicit OPEN operation, specify UC (user control) in positions 71 and 72 of the file description specifications. (See Chapter 2, "File Description Specifications" for restrictions when using the UC entry.) If a file is opened by RPG/400 and later closed by the CLOSE operation in the program, the programmer can reopen the file with the OPEN operation and the UC entry is not required in positions 71 and 72. When UC is not specified, RPG/400 opens the file at program initialization. If an OPEN operation is specified for a file that is already open, an error occurs.

Multiple OPEN operations in a program to the same file are valid as long as the file is closed when the OPEN operation is issued to it.

When you open a file with the ID option specified (on the file description specifications continuation line), the ID field is set to blanks. See the description of the ID option, in Chapter 2, "File Description Specifications."

Figure 9-29 on page 9-108 shows the OPEN operation used with the CLOSE operation.

ORxx (Or)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Blank	Required	Required	Blank	Blank	Blank	Blank

The ORxx operation is optional with the DOUxx, DOWxx, IFxx, and ANDxx operations. ORxx is specified immediately following a DOUxx, DOWxx, IFxx, ANDxx or ORxx statement. The ORxx operation allows a more complex condition to be specified for the DOUxx, DOWxx, and IFxx operations.

The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement within the appropriate section of the program. The control level entry must be the same as the control level entry for the associated DOUxx, DOWxx, or IFxx operation. Conditioning indicator entries (positions 9 through 17) are not permitted.

Factor 1 and factor 2 must contain a character literal, a numeric literal, or a field name. Factor 1 and factor 2 must be either both character data or both numeric data. The comparison of factor 1 and factor 2 follows the same rules as those given for the compare operations. See "Compare Operations" on page 9-9.

Figure 9-12 on page 9-51 shows an example of ORxx and ANDxx operations with a DOUxx operation.

OUT (Write a Data Area)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Required	Blank	Blank	Opt: ER	Blank

The OUT operation updates the data area specified in factor 2. To specify a data area in factor 2 of an OUT operation, you must ensure two things:

- The data area must also be specified in the result field of an *NAMVAR DEFN statement
- The data area must have been locked previously by an *LOCK IN statement or it must have been specified as a data area data structure by a U in position 18 of the input specifications (RPG/400 implicitly retrieves and locks data area data structures at program initialization).

Factor 1 can contain the reserved word *LOCK or can be blank. When factor 1 contains *LOCK, the data area remains locked after it is updated. When factor 1 is blank, the data area is unlocked after it is updated.

Factor 1 must be blank when factor 2 contains the name of the local data area.

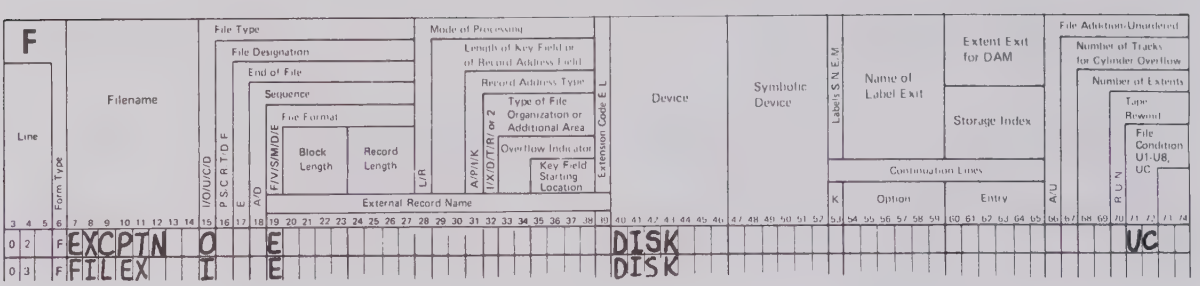
Factor 2 must be the name of the area to be updated or must be the reserved word *NAMVAR. When *NAMVAR is specified, all data areas defined in the program are updated. If an error occurs when one or more data areas are updated (for example, if you specify an OUT operation to a data area that has not been locked by the program), an error occurs on the OUT operation and the RPG/400 exception/error handling routine receives control. If a program status data structure (*PSSR) is specified, the data structure contains information on the data area in error. If a message is issued to the requester, the message identifies the data area in error.

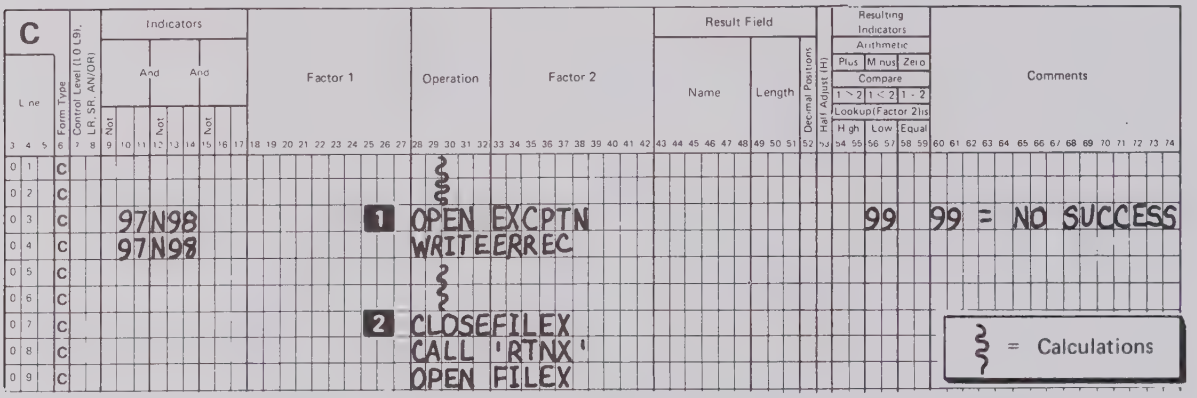
A resulting indicator can be specified in positions 56 and 57 to be set on if an error occurs during the operation. Positions 54-55 and 58-59 must be blank.

For further rules for the OUT operation, see "Data Area Operations" on page 9-15.

See Figure 9-21 on page 9-78 for an example of the OUT operation.

File Description Specifications





1 The explicit OPEN operation (line 03 of the calculation specifications) opens the EXCPTN file for processing if indicator 97 is on and indicator 98 is off. Note that the EXCPTN file on the file description specifications has UC specified in positions 71 and 72.

2 RPG/400 opens FILEX at program initialization. The explicit CLOSE operation closes FILEX before control is passed to RTNX. RTNX or another program can open and use FILEX. Upon return, the OPEN operation reopens the file. Because RPG/400 opens FILEX at program initialization, UC is not specified in positions 71 and 72 of the file description specifications.

Figure 9-29. OPEN Operation with CLOSE Operation

PARM (Identify Parameters)

Indicators				Result	Resulting Indicators		
7-8	9-17	Factor 1	Factor 2	Field	54-55	56-57	58-59
Opt	Blank	Optional	Optional	Required	Blank	Blank	Blank

The declarative PARM operation defines the parameters that compose a parameter list (PLIST). PARM operations can appear anywhere in calculations as long as they immediately follow the PLIST or CALL operation to which they refer. PARM statements must be in the order expected by the called program. One PARM statement, or as many PARM statements as are needed, can follow a PLIST or CALL.

The PARM operation can be specified anywhere within calculations, including total calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement in the appropriate section of the program. Conditioning indicator entries (positions 9 through 17) are not permitted.

Factor 1 and factor 2 entries are optional. If they are specified, the entries must be the same type (character or numeric) as specified in the result field. A literal cannot be specified in factor 1. Factor 1 and factor 2 must be blank if the result field contains the name of a multiple occurrence data structure.

The result field must contain the name of a field, data structure, or array that is to be the parameter. Also, the result field of a non *ENTRY PLIST can contain an array element. The result field can be numeric or character. The result field cannot contain *IN, *INxx, *IN,xx, a label, a literal, a data area name, a data area data structure name, or a table name. In addition, an array element, a data structure subfield name, the name of a compile-time array and the name of a program status or file information data structure (INFDS) are not allowed in the result field of PARM specified for an *ENTRY PLIST. A field name can be specified only once in an *ENTRY PLIST.

If an array is specified in the result field, the area defined for the array is passed to the called program. When a data structure with multiple occurrences is passed to the called program, all occurrences of the data structure are passed as a single field. However, if a subfield of a multiple occurrence data structure is specified in the result field, only the current occurrence of the subfield is passed to the called program.

There is only one storage location for each parameter field. It is in the calling program. What is passed to the called program on a PARM operation is the address of the storage location of the result field. If the called program changes the value of a parameter, it changes the data at that storage location. Therefore, when control returns to the calling program, the value of the parameter in the calling program (that is, the result field) has changed. Even if the called program ends in error after

it changes the value of a parameter, the changed value exists in the calling program. To preserve for later use the information passed to the called program, specify in factor 2 the name of the field that contains the information you want to pass to the called program. Factor 2 is copied into the result field, and the storage address of the result field is passed to the called program.

Because the program accesses the parameter fields by address, not field name, the calling and called parameters do not have to use the same field names for fields that are passed. The attributes of the corresponding parameter fields in the calling and called programs must be the same. If they are not, undesirable results will occur.

Upon running of a CALL operation the following occurs:

1. In the calling program, the contents of the factor 2 field of a PARM operation are copied into the result field (receiver field) of the same PARM operation.
2. In the called program, after it receives control and after any normal program initialization, the contents of the result field of a PARM operation are copied into the factor 1 field (receiver field) of the same PARM operation.
3. In the called program, when control is returned to the calling program, the contents of the factor 2 field of a PARM operation are copied into the result field (receiver field) of the same PARM operation. This move does not occur if the called program ends abnormally.
4. Upon return to the calling program, the contents of the result field of a PARM operation in the calling program are copied into the factor 1 field (receiver field) of the same PARM operation. This move does not occur if the called program ends abnormally or if an error occurs on the CALL operation.

Note: If the receiver field is numeric, the value of the field to be placed in the receiver field is added to a field of zeros and the sum is placed in the receiver field. If the receiver field is character, the field to be placed in the receiver field is moved (left-adjusted) into the receiver field, and the receiver field is padded with blanks.

To pass parameters to an RPG/400 program that is run through CL, the parameters can be specified on the command that calls the program. For a discussion of how to call and pass parameters to a program through CL, see the *Programming: Control Language Programmer's Guide.*

Figure 9-30 on page 9-112 illustrates the PARM operation.

PLIST (Identify a Parameter List)

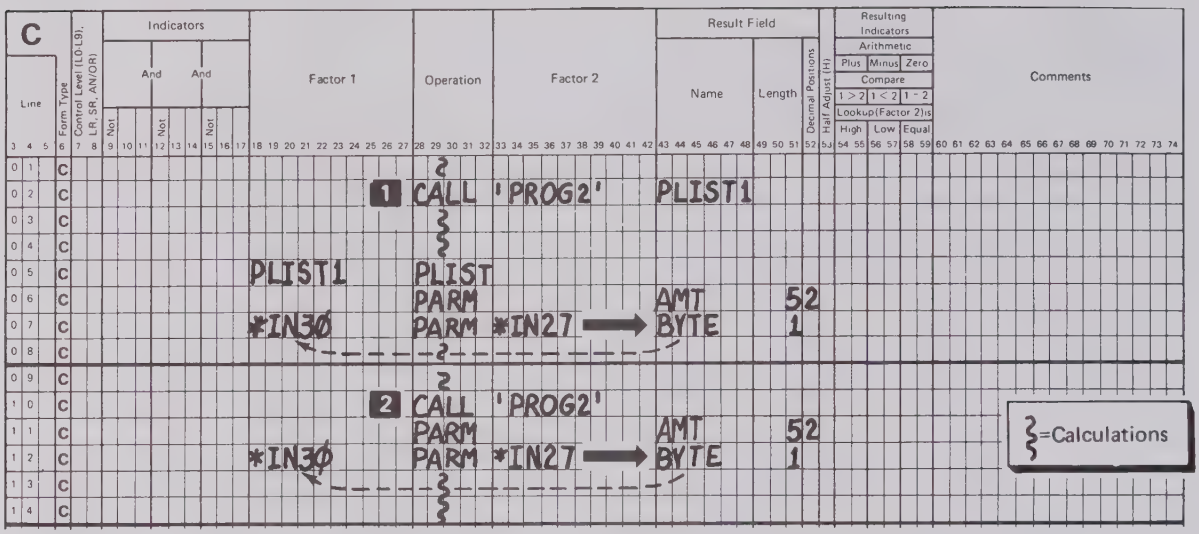
Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Blank	Required	Blank	Blank	Blank	Blank	Blank

The declarative PLIST operation defines a unique symbolic name for a parameter list to be specified in a CALL operation.

A PLIST operation can be specified anywhere within calculations, including within total calculations and between subroutines. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, an LR indicator, or an L0 entry to group the statement in the appropriate section of the program. Conditioning indicator entries (positions 9 through 17) are not permitted.

Factor 1 must contain the name of the parameter list. If the parameter list is the entry parameter list of a called program, factor 1 must contain *ENTRY. Only one *ENTRY parameter list can occur in a program. A parameter list is ended when an operation other than PARM is encountered.

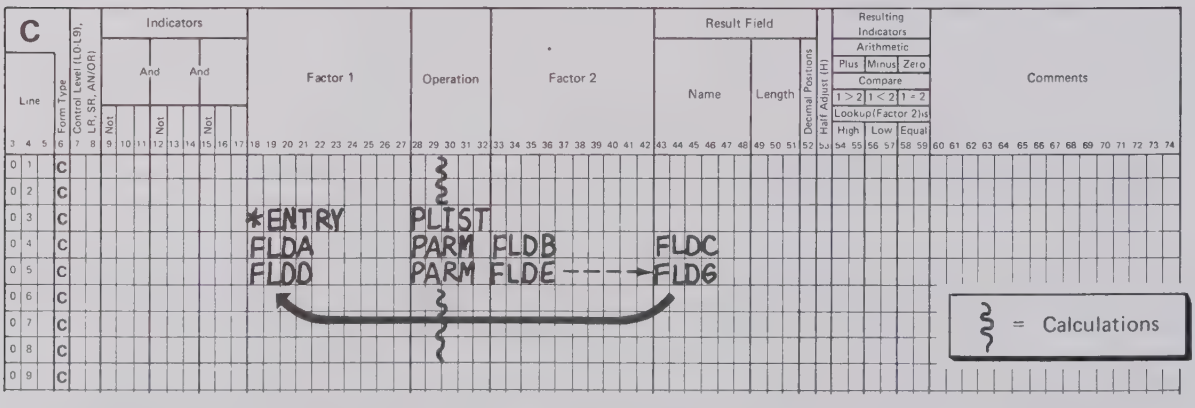
Figure 9-30 on page 9-112 illustrates the PLIST operation.



1 The CALL operation (line 02) calls PROG2 and allows PROG2 to access the data in the parameter list fields (lines 06 and 07). In line 07, when CALL is processed, the contents of the factor 2 field, *IN27, are placed in the result field, BYTE. When PROG2 returns control, the contents of the result field, BYTE, are placed in the factor 1 field, *IN30. Note that factor 1 and factor 2 entries on a PARM operation are optional (line 06).

2 This example is similar to **1** except that the PARM operations (lines 11 and 12) immediately follow a CALL operation instead of a PLIST operation.

Figure 9-30 (Part 1 of 2). PLIST/PARM Operations



This example shows the called program, PROG2. *ENTRY in factor 1 of the PLIST statement identifies this parameter list as the entry parameter list of the called program. When control is transferred to PROG2, the contents of the result fields (FLDC and FLDG) of the parameter list are placed in the factor 1 fields (FLDA and FLDD). When the called program returns, the contents of the factor 2 fields of the parameter list (FLDB and FLDE) are placed in the result fields (FLDC and FLDG). All of the fields are defined elsewhere in the program.

Figure 9-30 (Part 2 of 2). PLIST/PARM Operations

POST (Post)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Opt	Opt	Opt	Blank	Opt: ER	Blank

The POST operation puts information in an INFDS (file information data structure). The information is either information on the status of a specific program device or I/O feedback information associated with a file.

In factor 1, you may specify a program device name to get information about that specific program device. To specify the program device name, use either a character field of length 10 or less, or else a character literal. If you leave factor 1 blank, you get I/O feedback information.

In factor 2, specify the name of a file. Information for this file is posted in the INFDS associated with this file. If you specify a program device in factor 1, the file must be defined as a WORKSTN file.

If you specify a file in factor 2, you may leave the result field blank. RPG/400 will use the INFDS associated with this file. You may specify a file in factor 2 and its associated INFDS in the result field. If you leave factor 2 blank, you must specify an INFDS in the result field; RPG/400 will post information from the file associated with this INFDS.

In positions 56 and 57, you may specify an indicator that is set on if there is an error. If no indicator is specified, control passes to your INFSR subroutine (if you have specified one) or the default error/exception handler when an error/exception occurs.

Note: Even when a POST operation code is not processed, the existence of the code in your program can affect the way RPG/400 operates. Usually, the INFDS is updated at each input and output operation. However, if anywhere in your program, you have specified a POST operation with factor 1 blank, then RPG/400 updates the I/O Feedback Information area and the Device Dependent Feedback Information area in the INFDS of any file only when you process a POST operation for that file (except for the area *STATUS which is always updated). To ensure that the DUMP operation code provides up-to-date information in this case, issue a POST operation before the DUMP operation.

If a program has no POST operation code, or if it has only POST operation codes with factor 1 specified, then the INFDS is updated with each input-output operation or block of operations.

When a POST operation is processed, the associated file must be open. If you specify a program device on the POST operation, it does not have to be acquired by the file.

READ (Read a Record)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Optional	Blank	Opt: ER	R: EOF

The READ operation reads the record, currently being pointed to, from a full procedural file (identified by an F in position 16 of the file description specifications).

Factor 2 must contain the name of a file or record format to be read. A record format name in factor 2 is allowed only with an externally described file (E in position 19 of the file description specifications). It may be the case that a READ-by-format-name operation will receive a different format than the one you specified in factor 2. If so, your READ operation ends in error.

For further information on this case, refer to the *Communications Programmer's Guide* for more information on format selection processing for an ICF file. If you are using a BSC, CMN, or MXD file refer to the System/38 CL Reference Manual for information on the FMTSLT parameter on the CRTBSCF, CRTCMNF, or ADDCMNDEVE command respectively.

The result field can contain the name of a data structure into which the record is read only if the file named in factor 2 is a program described file (identified by an F in position 19 of the file description specifications). See "File Operations" on page 9-13 for information on how data is transferred between the file and the data structure.

If a READ operation is successful, the file is positioned at the next record that satisfies the read. If a READ operation is not successful, you must reposition the file (by a CHAIN, SETLL, or SETGT operation). An indicator can be specified in positions 56 and 57 to be set on if the READ operation does not complete successfully. If an error occurs when no indicator is specified, control passes to your INFSR subroutine (if specified) or the default error/exception handler.

An indicator must be specified in positions 58 and 59 to signal whether end of file occurred on the READ operation. The file must be repositioned after the indicator is set on in order to process any further successful sequential operation (for example, READ or READP) to the file. This indicator is set on or off every time READ runs.

Figure 9-31 on page 9-116 illustrates the READ operation.

When you specify a multiple device file in factor 2, the READ operation does one of the following:

- Reads data from the device specified in the most recent NEXT operation (if such a NEXT operation has been processed).

- Accepts the first response from any device that has been acquired for the file, and which was specified for "invite status" with the DDS keyword INVITE. If there are no invited devices, the operation receives an end-of-file. The input is processed according to the corresponding format. If the device is a work station, the last format written to the work station is used. If the device is a communications device, you may select the format.

  For further information on this case, refer to the *Communications Programmer's Guide* for more information on format selection processing for an ICF file. If you are using a BSC, CMN, or MXD file refer to the System/38 CL Reference Manual for information on the FMTSLT parameter on the CRTBSCF, CRTCMNF, or ADDCMNDEVE command respectively.

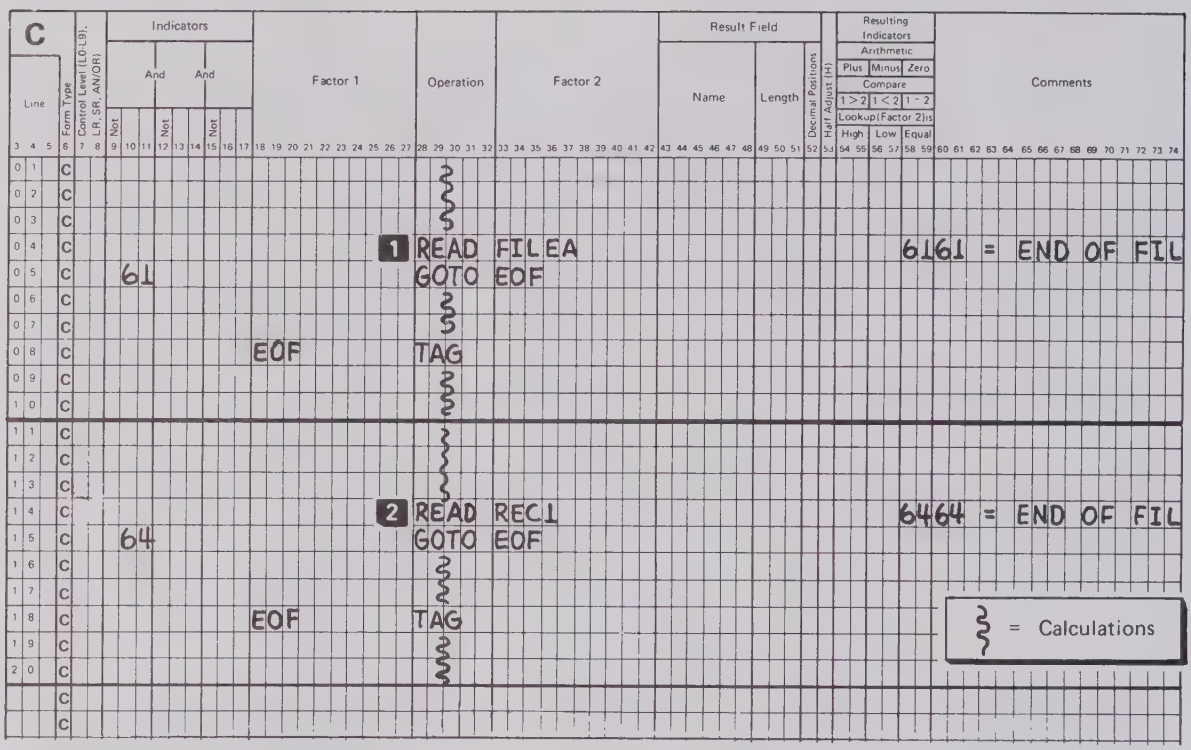
  The READ operation will stop waiting after a period of time in which no input is provided, or when one of the following CL commands has been entered with the controlled option specified:

  - ENDJOB (End Job)
  - ENDSBS (End Subsystem)
  - PWRDWNSYS (Power Down System)
  - ENDSYS (End System)

  The error indicator specified in positions 56 and 57 is set on. See the *Communications: Programmer's Guide* for a discussion of the WAITRCD parameter on the commands to create or modify a file. This parameter controls the length of time the READ operation waits for input.

When you specify a format name in factor 2, and the format name is associated with a multiple device file, data is read from the device identified by the field specified in the ID entry on the file continuation specifications. If there is no such entry, data is read from the device used in the last successful input operation.

READ (Read a Record)



1 READ (line 04) retrieves the next record from the file FILEA, which must be a full procedural file. Indicator 61 is set on if end of file occurs on READ, or if end of file has occurred previously and the file has not been repositioned. When indicator 61 is set on, the program branches to the label (EOF) specified in the GOTO statement (line 05).

2 READ (line 14) retrieves the next record of the type REC1 (factor 2) from an externally described file. (REC1 is a record format name.) Indicator 64 is set on if end of file occurs on READ, or if end of file has occurred previously and the file has not been repositioned. When indicator 64 is set on, the program branches to the label (EOF) specified in the GOTO statement (line 15).

Figure 9-31. READ Operations

READC (Read Next Modified Record)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Opt: ER	R: EOF

The READC operation can be used only with an externally described WORKSTN file to obtain the next changed record in a subfile. Factor 2 is required and must be the name of a record format defined as a subfile by the SFILE keyword on the file description specifications (see Chapter 2, "File Description Specifications" for information on the SFILE keyword).

For a multiple device file, data is read from the subfile record associated with a program device; the program device is identified by the field specified in the ID entry (on the file specifications continuation line). If there is no such entry, data is read from the program device used for the last successful input operation.

A resulting indicator can be specified in positions 56 and 57 to be set on if an error occurs while the operation is running. A resulting indicator in positions 58 and 59 is required; it is set on when there are no more changed records in the subfile.

READE (Read Equal Key)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Optional	Blank	Opt: ER	R: EOF

The READE operation retrieves the next sequential record from a full procedural file (identified by an F in position 16 of the file description specifications) if the key of the record matches the search argument specified in factor 1. If the key of the record does not match the search argument, the indicator that must be specified in positions 58 and 59 is set on, and the record is *not* returned to the program.

Factor 1, the search argument, is required and identifies the record to be retrieved. Factor 1 can be a field name or a literal. In addition, a KLIST name can be specified in factor 1 for an externally described file.

Factor 2 must contain the name of the file or record format to be retrieved. A record format name in factor 2 is allowed only with an externally described file (identified by an E in position 19 of the file description specifications).

The result field can contain the name of a data structure into which the record is read only if the file named in factor 2 is a program described file (identified by an F in position 19 of the file description specifications). See "File Operations" on page 9-13 for how data is transferred between the file and data structure.

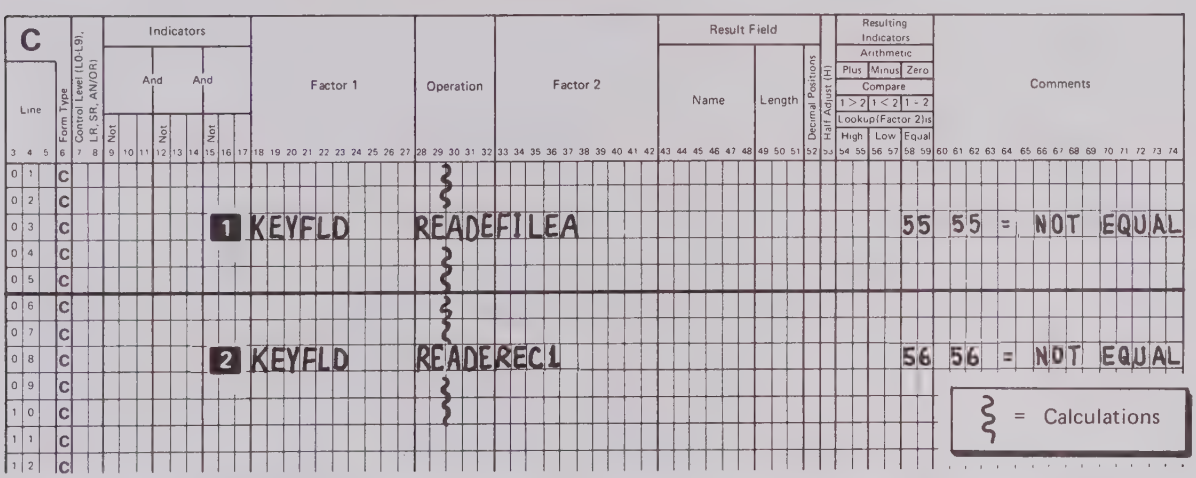
A resulting indicator can be specified in positions 56 and 57 to be set on if the operation does not complete successfully. A resulting indicator must be specified in positions 58 and 59. The indicator is set on if a record is not found with a key equal to the search argument or if end of file occurs. If a READE operation is not successful, you must reposition the file (for example, by a CHAIN or SETLL operation).

If a READE on a file defined for update is specified, RPG/400 requests a temporary lock on the next record before comparing the search argument to the key of that record. If the record is already locked, however, the program waits until the record is available before requesting the temporary lock and making the comparison. If the comparison is unequal, the record not found indicator is turned on, and the temporary record lock, which was placed on the record with the unequal key, is removed.

If one or more key fields in the file being read from are defined with ALTSEQ, ABSVAL, DIGIT, or ZONE the result of the read operation may differ from that which is expected. This is because the content of the field on which the access path is built may not be the same as the content that the READE operation is using for its internal comparison.

Note: A READE that immediately follows an OPEN operation retrieves the first record in the file if the key of the record matches the search argument.

Figure 9-32 on page 9-119 shows READE operations with a file name and record format name specified in factor 2.



1 The READE operation (line 03) retrieves the next record from the file FILEA and compares the key of the record read to the search argument, KEYFLD. Indicator 55 is set on if KEYFLD is not equal to the key of the record read or if end of file is encountered.

2 The READE operation (line 08) retrieves the next record of the type REC1 from an externally described file and compares the key of the record read to the search argument, KEYFLD. (REC1 is a record format name.) Indicator 56 is set on if KEYFLD is not equal to the key of the record read or if end of file is encountered.

Figure 9-32. READE Operations

READP (Read Prior Record)

Indicators				Result	Resulting Indicators		
7-8	9-17	Factor 1	Factor 2	Field	54-55	56-57	58-59
Opt	Opt	Blank	Required	Optional	Blank	Opt: ER	R: BOF

The READP operation reads the prior record from a full procedural file (identified by an F in position 16 of the file description specifications).

Factor 2 must contain the name of a file or record format to be read. A record format name in factor 2 is allowed only with an externally described file. If a record format name is specified in factor 2, the record retrieved is the first prior record of the specified type. Intervening records are bypassed.

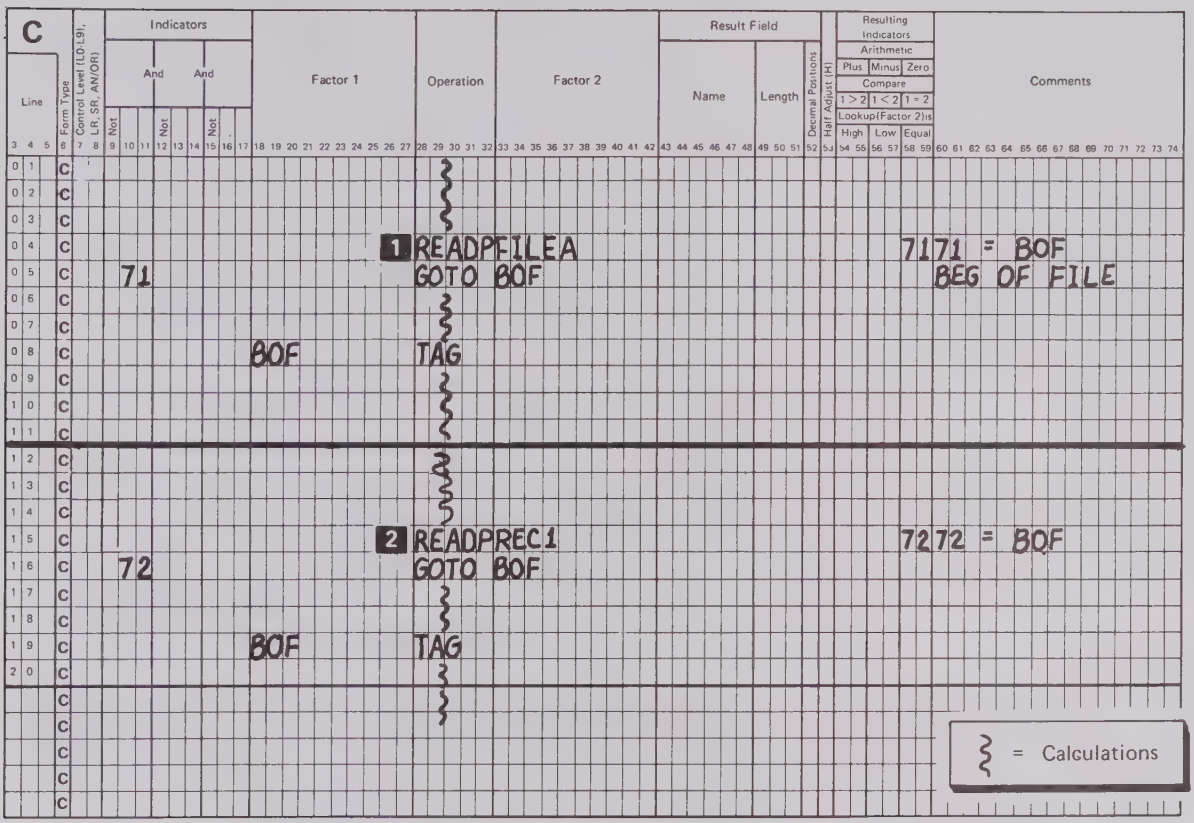
The result field can contain the name of a data structure into which the record is read only if the file named in factor 2 is a program described file (identified by an F in position 19 of the file description specifications). See "File Operations" on page 9-13 for how data is transferred between the file and data structure.

If a READP operation is successful, the file is positioned at the next record that satisfies the read. If a READP operation is not successful, you must reposition the file (for example, by a CHAIN or SETLL operation). An indicator can be specified in positions 56 and 57 to be set on if the READP operation does not complete successfully.

An indicator must be specified in positions 58 and 59 to be set on when no prior records exist in the file (beginning of file condition). If the file is not repositioned after this indicator is set on, the indicator is set for every subsequent READP operation to the file.

The file must be repositioned after the indicator is set on in order to process any further successful sequential operation (for example, READ or READP) to the file.

Figure 9-33 on page 9-121 shows READP operations with a file name and record format name specified in factor 2.



1 The READP operation (line 04) reads the prior record from the file, FILEA. Indicator 71 is set on if beginning of file is encountered while the READP operation is running. When indicator 71 is set on, the program branches to the label BOF specified in the GOTO operation (line 05).

2 The READP operation (line 15) reads the next prior record of the type REC1 from an externally described file. (REC1 is a record format name). Indicator 72 is set on if beginning of file is encountered during processing of the READP operation. When indicator 72 is set on, the program branches to the label BOF specified in the GOTO operation (line 16).

Figure 9-33. READP Operations

REL (Release)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Blank	Blank	Opt: ER	Blank

The REL operation releases the program device specified in factor 1 from the multiple device WORKSTN file specified in factor 2.

In factor 1, specify the program device name. Use either a character field of length 10 or less, or a character literal. In factor 2, specify the file name.

In positions 56 and 57, you may specify an indicator that is set on when an error occurs. If no indicator is specified, and an error occurs, control passes to your INFSR subroutine (if specified) or the default error/exception handler.

When there are no program devices acquired to a WORKSTN file, the next READ-by-file-name or cycle-read gets an end-of-file condition. It is your responsibility to decide what the program does next.

The REL operation is valid only for multiple device WORKSTN files.

RETRN (Return to Caller)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Blank	Blank	Blank	Blank	Blank

The RETRN operation causes a return to the caller.

The return to the caller functions as follows:

1. RPG/400 checks the halt indicators. If a halt indicator is on, the program ends abnormally (all open files are closed, an error return code is set to indicate to the caller that the program has ended abnormally, and control returns to the caller).
2. If no halt indicators are on, RPG/400 checks the LR indicator. If LR is on, the program ends normally.
3. If no halt indicator is on and LR is not on, the program returns to the caller.

ROLBK (Roll Back)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Blank	Blank	Blank	Opt: ER	Blank

The ROLBK operation:

- Eliminates all the changes to your files that have been specified in output operations since the previous COMIT or ROLBK operation (or since the beginning of operations under commitment control if there has been no previous COMIT or ROLBK operation).
- Releases all the record locks for the files you have under commitment control.
- Repositions the file to its position at the time of the previous COMIT operation (or at the time of the file OPEN, if there has been no previous COMIT operation.)

The file changes and the record-lock releases apply to all the files under commitment control in your routing step, whether the changes have been requested by the program issuing the ROLBK operation, or by another program in the same routing step. It is not necessary for the program issuing the ROLBK operation to have any files under commitment control. For example, suppose program A calls program B and program C. Program B has files under commitment control and program C does not. A ROLBK operation in program C still affects the files changed by program B.

The optional indicator in positions 56 and 57 is set on if the operation is not successfully completed.

SETGT (Set Greater Than)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Blank	Opt: NR	Opt: ER	Blank

The SETGT operation positions a file at the next record with a key or relative record number that is greater than the key or relative record number specified in factor 1. The file must be a full procedural file (identified by an F in position 16 of the file description specifications).

Factor 1 is required. If the file is accessed by key, factor 1 can be a field name or a literal that is used as the search argument in positioning a file. In addition, a KLIST name can be specified in factor 1 for an externally described file that is positioned by key. If the file is accessed by relative record number, factor 1 must be an integer literal or a numeric field with zero decimal positions.

Factor 2 is required and can be either a file name or a record format name. A record format name in factor 2 is allowed only with an externally described file.

A resulting indicator in positions 54 and 55 can be specified to be set on if no record is found with a key or relative record number that is greater than the search argument specified in factor 1. Any valid resulting indicator can be specified in positions 56 and 57 to be set on if an error occurs during processing of the operation.

If the SETGT operation is not successful (no-records-found condition), the file is positioned to the end of the file.

Figurative constants can also be used to position the file. When used with a file with a composite key, figurative constants are treated as though each field of the key contained the figurative constant value. In most cases, *LOVAL positions the file so that the first read retrieves the record with the lowest key. In most cases, *HIVAL positions the file so that a READ receives an end-of-file indication; a subsequent READP retrieves the last record in the file. However, note the following cases for using *LOVAL and *HIVAL:.

- With an externally described file that has a key in descending order, *HIVAL positions the file so that the first read operation retrieves the first record in the file (the record with the highest key), and *LOVAL positions the file so that a READP operation retrieves the last record in the file (the record with the lowest key).
- If you or someone else adds a record or alters a key field after a SETGT operation with either *LOVAL or *HIVAL, the file may no longer be positioned to the record with the lowest or highest key.

- *LOVAL represents a key value X'99...9D' and *HIVAL represents a key value X'99...9F'. When a program described file has a packed decimal key specified in the file specifications but the actual file key field contains character data, records may have keys that are less than *LOVAL or greater than *HIVAL. When a key field contains unsigned binary data, *LOVAL may not be the lowest key.

Following the SETGT operation, a file is positioned so that it is immediately before the first record whose key or relative record number is greater than the search argument specified in factor 1. You retrieve this record by reading the file. Before you read the file, however, records may be deleted from the file by another job or through another file in your job. Thus, you may not get the record you expected. For information on preventing unexpected modification of your files, see the discussion of allocating objects in the *Programming: Control Language Reference*.

Figure 9-34 on page 9-127 illustrates the SETGT operation.

```
C                                                
```

SETGT (Set Greater Than)

```
C                                                
```

SETLL (Set Lower Limit)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Required	Required	Blank	Opt: NR	Opt: ER	Opt: EQ

The SETLL operation positions a file at the next record that has a key or relative record number that is greater than or equal to the search argument (key or relative record number) specified in factor 1. The file must be a full procedural file (identified by an F in position 16 of the file description specifications).

Factor 1 is required. If the file is accessed by key, factor 1 can be a field name or a literal that is used as the search argument in positioning the file. In addition, a KLIST name can be specified in factor 1 for an externally described file that is positioned by key. If the file is accessed by relative record number, factor 1 must be an integer literal or a numeric field with zero decimal positions.

Factor 2 is required and can contain either a file name or a record format name. A record format name in factor 2 is allowed only with an externally described file.

The resulting indicators reflect the status of the operation. If an indicator is specified in positions 54 and 55, it is set on when the search argument is greater than the highest key or relative record number in the file. If an indicator is specified in positions 56 and 57, it is set on when an error occurs during running of the operation. If an indicator is specified in positions 58 and 59, it is set on when a record is present whose key or relative record number is equal to the search argument.

If factor 2 contains a file name for which the lower limit is to be set, the file is positioned at the first record with a key or relative record number equal to or greater than the search argument specified in factor 1.

If factor 2 contains a record format name for which the lower limit is to be set, the file is positioned at the first record of the specified type that has a key equal to or greater than the search argument specified in factor 1.

Figurative constants can be used to position the file. When used with a file with a composite key, figurative constants are treated as though each field of the key contained the figurative constant value. In most cases, *LOVAL positions the file so that the first read retrieves the record with the lowest key. In most cases, *HIVAL positions the file so that a READP retrieves the last record in the file, or a READ receives an end-of-file indication. However, note the following cases for using *LOVAL and *HIVAL:.

- With an externally described file that has a key in descending order, *HIVAL positions the file so that the first read operation retrieves the first record in the file (the record with the highest key), and *LOVAL positions the file so that a

READP operation retrieves the last record in the file (the record with the lowest key).

- If a record is added or a key field altered after a SETLL operation with either *LOVAL or *HIVAL, the file may no longer be positioned to the record with the lowest or highest key.
- *LOVAL represents a key value X'99...9D' and *HIVAL represents a key value X'99...9F'. When a program described file has a packed decimal key specified in the file specifications but the actual file key field contains character data, records may have keys that are less than *LOVAL or greater than *HIVAL. When a key field contains unsigned binary data, *LOVAL may not be the lowest key.

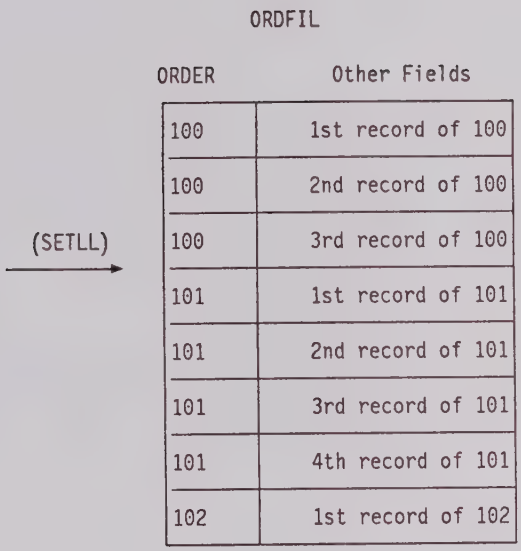
Figure 9-34 on page 9-127 (part 2 of 2) shows the use of figurative constants with the SETGT operation. Figurative constants are used the same way with the SETLL operation.

Remember the following when using the SETLL operation:

- If the SETLL operation is not successful (no records found condition), the file is positioned to the end of the file.
- When end of file is reached on a file being processed by SETLL, another SETLL can be issued to reposition the file.
- After a SETLL operation successfully positions the file at a record, you retrieve this record by reading the file. Before you read the file, however, records may be deleted from the file by another job or through another file in your job. Thus, you may not get the record you expected. Even if the resulting indicator in positions 58 and 59 is set on to indicate you found a matching record, you may not get that record. For information on preventing unexpected modification of your files, see the discussion of allocating objects in the *Programming: Control Language Reference*.

Figure 9-35 on page 9-132 illustrates the SETLL operation.

In this example, the file ORDFIL contains order records. The key field is the order number (ORDER) field. There are multiple records for each order. ORDFIL looks like this:



The calculation specifications are:

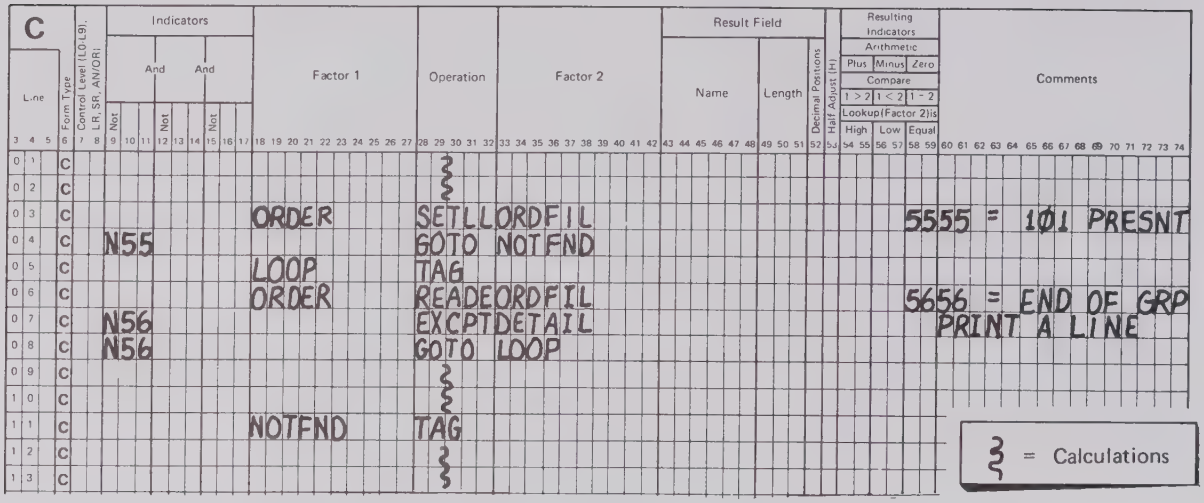


Figure 9-35. SETLL Operation

All the 101 records in ORDFIL are to be printed. The value 101 has previously been placed in ORDER. The SETLL operation (line 03) positions the file at the first 101 record. Because there are 101 records, indicator 55 is set on and line 04 is not processed. The READE operation (line 06) reads the first 101 record. Because indicator 56 is not on, the lines conditioned by N56 are processed. The EXCPT operation is processed (line 07), and the program branches to the label specified in the GOTO operation (line 05). The READE operation reads the second, third, and fourth 101 records in the same manner as the first 101 record was read. After the fourth 101 record is read, the READE operation is again attempted. Because the 102 record is not of the same group, indicator 56 is set on and lines 07 and 08 are bypassed.

SETOF (Set Off)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Blank	Blank	1 required		

The SETOF operation sets off any indicators specified in positions 54 through 59. At least one resulting indicator must be specified in positions 54 through 59. Entries of 1P and MR are not valid. Setting off L1 through L9 indicators does not automatically set off any lower control level indicators.

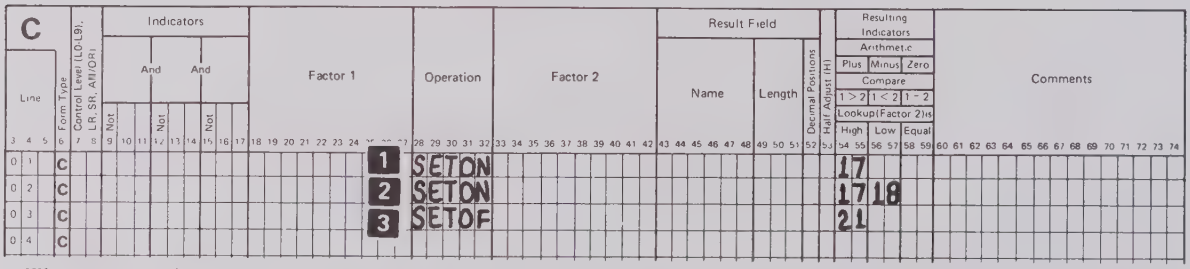
Figure 9-36 on page 9-134 illustrates the SETOF operation.

SETON (Set On)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Blank	Blank	1 required		

The SETON operation sets on any indicators specified in positions 54 through 59. At least one resulting indicator must be specified in positions 54 through 59. Entries of 1P, MR, KA through KN, and KP through KY are not valid. Setting on L1 through L9 indicators does not automatically set on any lower control level indicators.

Figure 9-36 illustrates the SETON operation.



The SETON and SETOF operations set from one to three indicators specified in positions 54 through 59 on and off. In these examples, the following occurs:

1. The SETON operation sets indicator 17 on.
2. The SETON operation sets indicators 17 and 18 on.
3. The SETOF operation sets indicator 21 off.

Figure 9-36. SETON and SETOF Operations

SHTDN (Shut Down)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Blank	Blank	Req	Blank	Blank

The SHTDN operation allows the programmer to determine whether the system operator has requested shutdown. If the system operator has requested shutdown, the resulting indicator specified in positions 54 and 55 is set on. Positions 54 and 55 must contain one of the following indicators: 01 through 99, L1 through L9, U1 through U8, H1 through H9, LR, or RT.

The system operator can request shutdown by specifying the *CNTRLD option on the following CL commands: ENDJOB (End Job), PWRDWNSYS (Power Down System), ENDSYS (End System), and ENDSBS (End Subsystem). For information on these commands see the *Programming: Control Language Reference*.

Positions 56 through 59 must be blank.

See Figure 9-37 for an example of the SHTDN operation.

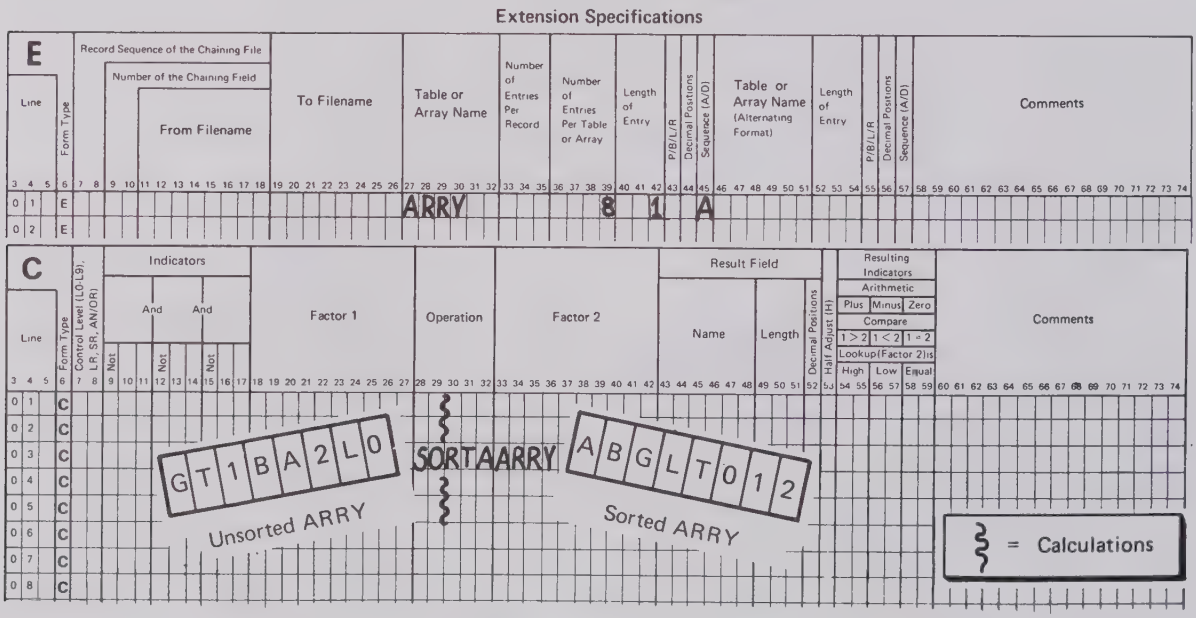
| C | | | |

SORTA (Sort an Array)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Blank	Blank

Factor 2 contains the name of an array to be sorted. The array is sorted into sequence (ascending or descending), depending on the sequence specified for the array in position 45 of the extension specifications. If no sequence is specified, the array is sorted into ascending sequence. The array *IN cannot be specified in factor 2 of a SORTA operation. A related array, such as a second array defined on the same extension specification, is not sorted. Only the array specified in factor 2 is sorted.

See Figure 9-38 for an example of the SORTA operation.



The SORTA operation sorts ARRY into ascending sequence because A is specified in position 45 of the extension specifications.

Figure 9-38. SORTA Operation

SQRT (Square Root)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Blank	Blank	Blank

The SQRT operation derives the square root of the field named in factor 2. The square root of factor 2 is placed in the result field.

An entire array can be used in a SQRT operation if factor 2 and the result field contain array names.

The number of decimal positions in the result field can be either less than or greater than the number of decimal positions in factor 2. However, the result field should not have less than half the number of decimal positions in factor 2.

If the value of the factor 2 field is zero, the result field value is also zero. If the value of the factor 2 field is negative, the RPG/400 exception/error handling routine receives control.

For further rules on the SQRT operation, see "Arithmetic Operations" on page 9-5.

See Figure 9-1 on page 9-7 for an example of the SQRT operation.

SUB (Subtract)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Optional	Required	Required	Opt: +	Opt: –	Opt: Z

If factor 1 is specified, factor 2 is subtracted from factor 1 and the difference is placed in the result field. If factor 1 is not specified, the contents of factor 2 are subtracted from the contents of the result field.

For rules for the SUB operation, see "Arithmetic Operations" on page 9-5.

See Figure 9-1 on page 9-7 for examples of the SUB operation.

TAG (Tag)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Blank	Required	Blank	Blank	Blank	Blank	Blank

The declarative TAG operation names the label that identifies the destination of a GOTO or CABxx operation.

The TAG operation can be specified anywhere within calculations, including within total calculations. The control level entry (positions 7 and 8) can be blank or can contain an L1 through L9 indicator, the LR indicator, or the L0 entry to group the statement within the appropriate section of the program. Conditioning indicator entries (positions 9 through 17) are not permitted.

Factor 1 must contain the name of the destination of a GOTO or CABxx operation. This name must be a unique symbolic name and must be specified in factor 2 of a GOTO operation or in the result field of a CABxx operation. The name can be used as a common point for multiple GOTO or CABxx operations.

See Figure 9-19 on page 9-74 for examples of the TAG operation.

TESTB (Test Bit)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	1 required		

The TESTB operation compares the bits identified in factor 2 with the corresponding bits in the field named as the result field. The result field must be a one-position character field. Resulting indicators in positions 54 through 59 reflect the status of the result field bits. Factor 2 is always a source of bits for the result field.

Factor 2 can contain:

- *Bit numbers 0-7:* From 1 to 8 bits can be tested per operation. The bits to be tested are identified by the numbers 0 through 7 (0 is the leftmost bit). The bit numbers must be enclosed in apostrophes, and the entry must begin in position 33. For example, to test bits 0, 2, and 5, enter '025' in factor 2.
- *Field name:* The name of a one-position character field, table element, or array element can be specified in factor 2. In this case, the bits that are on in the field, table element, or array element are compared with the corresponding bits in the result field; bits that are off are not affected. The field specified in factor 2 or in the result field can be an array element if each element of the array is a one-position character field.

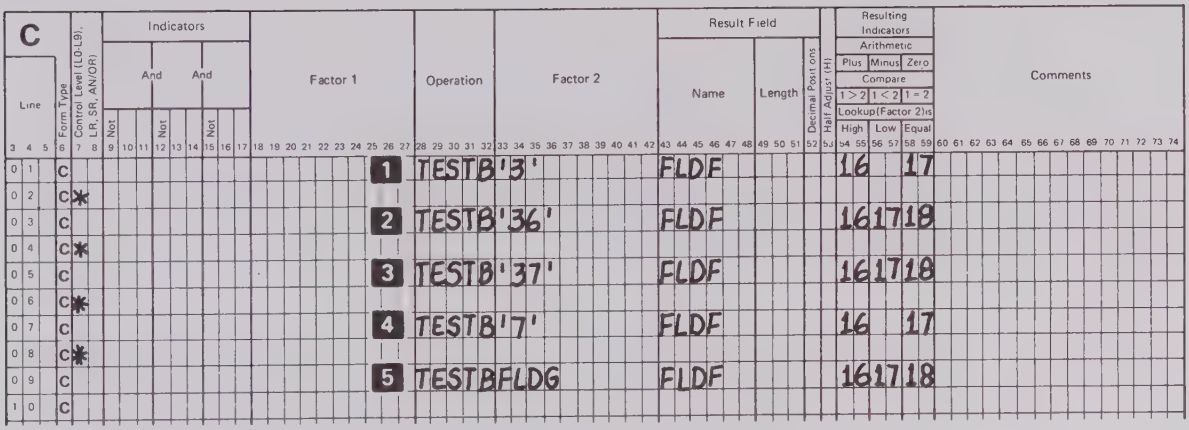
Figure 9-39 on page 9-141 illustrates uses of the TESTB operation.

Indicators assigned in positions 54 through 59 reflect the status of the result field bits. At least one indicator must be assigned, and as many as three can be assigned for one operation. Two indicators can be the same for a TESTB operation, but not three. For TESTB operations, the resulting indicators are set on as follows:

- *Positions 54 and 55:* An indicator in these positions is set on if the bit numbers specified in factor 2 or each bit that is on in the factor 2 field is off in the result field. That is, all of the specified bits are equal to off.
- *Positions 56 and 57:* An indicator in these positions is set on if the bit numbers specified in factor 2 or the bits that are on in the factor 2 field are of mixed status (some on, some off) in the result field. That is, some of the specified bits are equal to on.

  Note: If only one bit is to be tested, these positions must be blank. If a field name is specified in factor 2 and it has only one bit on, an indicator in positions 56 and 57 is not set on.
- *Positions 58 and 59:* An indicator in these positions is set on if the bit numbers specified in the factor 2 or each bit that is on in factor 2 field is on in the result field. That is, all of the specified bits are equal to on.

  Note: If the field in factor 2 has no bits on, then no indicators are set on.



The field bit settings are:

```
FLDF = 00000001
FLDG = 11110001
```

As a result of the TESTB operations:

1. Indicator 16 is set on because bit 3 is off (0) in FLDF. Indicator 17 is set off.
2. Indicator 16 is set on because both bits 3 and 6 are off (0) in FLDF. Indicators 17 and 18 are set off.
3. Indicator 17 is set on because bit 3 is off (0) and bit 7 is on (1) in FLDF. Indicators 16 and 18 are set off.
4. Indicator 17 is set on because bit 7 is on (1) in FLDF. Indicator 16 is set off.
5. Indicator 17 is set on because bits 0, 1, 2, and 3 are off (0) and bit 7 is on (1). Indicators 16 and 18 are set off.

Figure 9-39. TESTB Operations

TESTN (Test Numeric)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Blank	Required	1 required		

The TESTN operation tests a character result field for the presence of zoned decimal digits and blanks. The result field must be a character field. To be considered numeric, each character in the field, except the low-order character, must contain a hexadecimal F zone and a digit (0 through 9). The low-order character is numeric if it contains a hexadecimal C, hexadecimal D, or hexadecimal F zone, and a digit (0 through 9). Note that the alphabetic characters A through R, should they appear in the low-order position of a field, are treated as negative numbers by TESTN. As a result of the test, resulting indicators are set on as follows:

- *Positions 54 and 55:* All the characters in the result field are numeric, or the result field is a one-character field containing a letter from A to R.
- *Positions 56 and 57:* The result field contains both numeric characters and at least one leading blank. For example, the values b123 or bb123 set this indicator on. However, the value b1b23 is an invalid numeric field because of the embedded blanks, so this value does not set this indicator on.

  Note: An indicator cannot be specified in positions 56 and 57 when a field of length one is tested because the character field must contain at least one numeric character and one leading blank.
- *Positions 58 and 59:* The result field contains all blanks.

The same indicator can be used for more than one condition. If any of the conditions exist, the indicator is set on.

To prevent undesirable results or an abnormal end of a program, the TESTN operation validates data in fields before arithmetic or editing operations are processed on the fields. Following validation, the field must be moved to a numeric field to process the arithmetic and editing operations.

Figure 9-40 on page 9-143 illustrates uses of the TESTN operation.

| C | | | | Indicators | | | |

TESTZ (Test Zone)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Blank	Required	1 required		

The TESTZ operation tests the zone of the leftmost character in the result field. The result field must be a character field. Resulting indicators are set on according to the results of the test. At least one resulting indicator must be specified in positions 54 through 59. The characters &, A through I, and any character with the same zone as the character A set the indicator in positions 54 and 55 on. The characters - (minus), J through R, and any character with the same zone as the character J set the indicator in positions 56 and 57 on. Characters with any other zone set the indicator in positions 58 and 59 on.

TIME (Time of Day)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Blank	Required	Blank	Blank	Blank

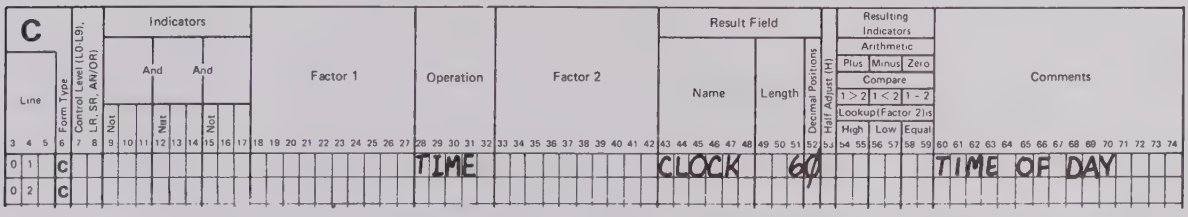
The TIME operation accesses the system time of day and, if specified, the system date at any time during program processing. The system time is based on the 24-hour time system.

The result field must specify the name of a 6- or 12-digit numeric field (zero decimal positions) into which the time of day or the time of day and the system date are written.

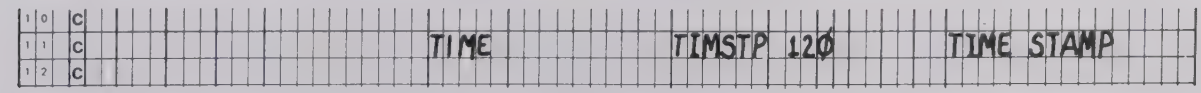
To access the time of day only, specify the result field as a 6-digit numeric field. To access both the time of day and the system date, specify the result field as a 12-digit numeric field. The time of day is always placed in the first six positions of the result field in the following format:

hhmmss (where hh = hours, mm = minutes, and ss = seconds)

If the system date is included, it is placed in positions 7 through 12 of the result field. The date format depends on the data format job attribute QDATFMT and can be mmddyy, ddmmyy, yymmdd, or Julian. The Julian format contains the year in positions 7 and 8, the day (1 through 366, right adjusted, with zeros in the unused high-order positions) in positions 9 through 11, and 0 in position 12.



When the TIME operation in line 01 is processed (with a 6-digit numeric field), the current time (in the form hhmmss) is placed in the result field CLOCK. The TIME operation is based on the 24-hour time system, for example, 132710. (In the 12-hour time system 132710 is 1:27:10 p.m.) CLOCK can then be specified in the output specifications.



When the TIME operation in line 11 is processed (with a 12-digit numeric field), the current time and day is placed in the result field TIMSTP. The first 6 digits are the time and the last 6 digits are the date. The TIME operation is based on the 24-hour time system, for example, 093315121579. (In the 12-hour time system 093315 is 9:33:15 a.m.) TIMSTP can then be specified in the output specifications.

Figure 9-41. TIME Operations

UNLCK (Unlock a Data Area)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Blank	Blank	Opt: ER	Blank

The UNLCK operation unlocks one or all locked data areas in a program. Before a data area can be used with the UNLCK operation, it must be specified in the result field of an *NAMVAR DEFN statement If the UNLCK operation is specified to a data area that is already unlocked, an error does not occur.

Factor 2 must contain the name of the data area to be unlocked or the reserved word *NAMVAR. When *NAMVAR is specified in factor 2, all data areas in the program that are locked are unlocked. Factor 2 must not refer to the local data area.

A resulting indicator can be specified in positions 56 and 57 to be set on if an error occurs on the operation. Positions 54, 55, 58, and 59 must be blank.

UPDAT (Modify Existing Record)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Optional	Blank	Opt: ER	Blank

The UPDAT operation modifies the last record retrieved for processing from an update file (identified by a U in position 15 of the file description specifications).

Factor 2 must contain the name of a file or record format to be updated. A record format name in factor 2 is required with an externally described file. The record format name must be the name of the last record read from the file; otherwise an error occurs. A file name in factor 2 is required with a program described file.

The result field must contain a data structure name if factor 2 contains a file name. The updated record is written directly from the data structure to the file. The result field must be blank if factor 2 contains a record format name.

A resulting indicator can be specified in positions 56 and 57 to be set on if the UPDAT operation does not complete successfully.

Remember the following when using the UPDAT operation:

- When a record format name is specified in factor 2, the current values in the program for the fields in the record definition are used to modify the record.
- If some but not all fields in a record are to be updated, use the output specifications and not the UPDAT operation.
- Before UPDAT is issued to a file or record, a valid input operation (READ, READC, READE, READP, CHAIN, or primary/secondary file) must be issued to the same file or record. If the read operation returns with an error condition or if a no-record-found condition occurs on a CHAIN operation, UPDAT cannot be issued.
- Consecutive UPDAT operations to the same file or record are not valid. Intervening successful read operations must be issued.
- Beware of using the UPDAT operation on primary or secondary files during total calculations. At this stage in the RPG/400 cycle, the fields from the current record (the record that is about to be processed) have not yet been moved to the processing area. Therefore, the UPDAT operation updates the current record with the fields from the preceding record. Also, when the fields from the current record are moved to the processing area, they are the fields that were updated from the preceding record.
- For multiple device files, specify a subfile record format in factor 2. The operation is processed for the program device identified in the field specified in the ID entry of the file specifications continuation line. (If there is no such entry, the device for the last successful input operation is used.) This device must be the

same device as you specified for the input operation that must precede the UPDAT operation. You must not process input or output operations to other devices in between the input and UPDAT operations. If you do, your UPDAT operation will fail.

WRITE (Create New Records)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Optional	Blank	Opt: ER	Opt: EOF

The WRITE operation writes a new record to a file.

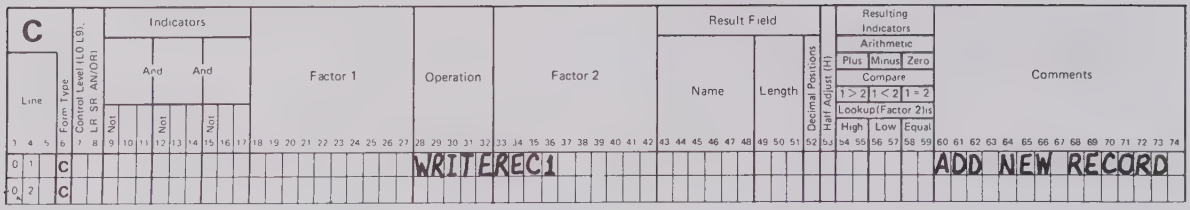
Factor 2 must contain the name of a file or record format. A record format name is required in factor 2 with an externally described file. A file name in factor 2 is required with a program described file, and the result field must contain the name of a data structure. The record is written directly from the data structure to the file. The result field must be blank if factor 2 contains a record format name.

Positions 56 and 57 can contain an indicator to be set on if the WRITE operation does not complete successfully. On a WRITE to a subfile (SFILE) record name, an indicator can be specified in positions 58 and 59; it is set on when the subfile is filled.

Remember the following when using the WRITE operation:

- When factor 2 contains a record format name, the current values in the program for all the fields in the record definition are used to construct the record.
- When records that use relative record numbers are written to a file, you must update the RECNO (relative record number) field so it contains the relative record number of the record to be written.
- When you use the WRITE operation to add records to a DISK file, you must specify an A in position 66 of the file description specifications (see "Position 66 (File Addition/Unordered)" on page 2-16).
- Device dependent functions are not available. For example, if a WRITE is issued to a PRINTER device, there is no spacing or skipping (normally specified in columns 17 through 22 of the output specifications). If the file is externally described, these functions are part of the external description.
- For a multiple device file, data is written to the program device named in the field specified in the ID entry on the file specifications continuation line. If there is no such entry, data is written to the program device for which the last successful input operation was processed.

See Figure 9-42 on page 9-150 for an example of the WRITE operation.



The WRITE operation writes all of the fields in the record format, REC1, to a file. The REC1 record is contained in an externally described file.

Figure 9-42. WRITE Operation

XFOOT (Summing the Elements of an Array)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Opt: +	Opt: –	Opt: Z

XFOOT adds the elements of an array together and places the sum into the field specified as the result field. Factor 2 contains the name of the array.

If half adjust (position 53) is specified, it occurs after all elements are summed and before the results are moved into the result field. If the result field is an element of the array specified in factor 2, the value of the element before the XFOOT operation is used to calculate the total of the array.

For further rules for the XFOOT operation, see "Arithmetic Operations" on page 9-5.

See Figure 9-1 on page 9-7 for an example of the XFOOT operation.

Z-ADD (Zero and Add)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Opt: +	Opt: –	Opt: Z

Factor 2 is added to a field of zeros. The sum is placed in the result field. Factor 1 is not used.

For the rules for the Z-ADD operation, see "Arithmetic Operations" on page 9-5.

See Figure 9-1 on page 9-7 for an example of the Z-ADD operation.

Z-SUB (Zero and Subtract)

Indicators				Result	Resulting Indicators		
7–8	9–17	Factor 1	Factor 2	Field	54–55	56–57	58–59
Opt	Opt	Blank	Required	Required	Opt: +	Opt: –	Opt: Z

Factor 2 is subtracted from a field of zeros. The difference, which is actually the negative of factor 2, is placed in the result field. The operation can be used to change the sign of a field. Factor 1 is not used.

For the rules for the Z-SUB operation see "Arithmetic Operations" on page 9-5.

See Figure 9-1 on page 9-7 for an example of the Z-SUB operation.

Chapter 10. RPG Words with Special Functions

RPG/400 provides the following reserved words that have special functions within a program:

- UDATE, UMONTH, UYEAR, and UDAY allow you to access a job-related date, or a portion of it, to be used in the program.
- PAGE, PAGE1-PAGE7 can be used for numbering the pages of a report, for record sequence numbering, or to sequentially number output fields.
- Figurative constants (*BLANK/*BLANKS, *ZERO/*ZEROS, *HIVAL, *LOVAL, and *ALL'X..') are implied literals that allow specifications without referring to length.
- *IN and *INxx allow indicators to be referred to as data.
- The following reserved words define symbolic locations within the file information data structure (INFDS) and the program status data structure.

  *ROUTINE

  *STATUS

  *FILE

  *RECORD

  *OPCODE

  *PARMS

  *PROGRAM

- The following reserved words provide symbolic labels for the ENDSR operation for the file and program exception/error subroutines.

*DETL	Detail lines
*GETIN	Get input record
*TOTC	Total calculations
*TOTL	Total lines
*OFL	Overflow lines
*DETC	Detail calculations
*CANCL	Cancel the program

- *PLACE allows repetitive placement of fields in an output record. (See Chapter 7, "Output Specifications" for more information on *PLACE.)
- *ALL allows all fields that are defined for an externally described file to be written on output. (See Chapter 7, "Output Specifications" for more information on *ALL)

User Date (UDATE, UMONTH, UDAY, UYEAR)

The user date special words (UDATE, UMONTH, UDAY, UYEAR) allow the programmer to supply a date for the program at run time. The user date special words access the job date that is specified in the job description. The user dates can be written out at output time, and are written in the format specified by the control specification. (For a description of the job date, see the *Programming: Work Management Guide*.)

Rules for User Date

Remember the following rules when using the user date:

- UDATE, when specified in positions 32 through 37 of the output specifications, prints a 6-character numeric date field in one of three formats:

  Month/day/year

  Year/month/day

  Day/month/year

  Use positions 19 and 20 of the control specification to specify the date format and the editing to be done. If positions 19 and 20 are blank, the date format is determined by the contents of position 21.

- UDATE, UMONTH, UDAY, and UYEAR are set when the job starts running and are not updated when a program runs over midnight. Use the TIME operation code to obtain the time and date while the program is running.

- UMONTH, UDAY, and UYEAR, when specified in positions 32 through 37 of the output specifications, print a 2-position numeric date field. Use UMONTH to print the month only, UDAY to print the day only, and UYEAR to print the year only.

- UDATE can be edited when it is written if the Y edit code is specified in position 38 of the output specifications. The control specification entry in position 20 determines the separator character to be inserted; for example, 12/31/79, 31.12.79.

- UMONTH, UDAY, and UYEAR cannot be edited by the Y edit code in position 38 of the output specifications.

- UDATE, UMONTH, UDAY, and UYEAR can be used in factor 1 or factor 2 of the calculation specifications.

- UDATE, UMONTH, UDAY, and UYEAR cannot appear as the result field in a calculation or as an input field.

- Blank-after (position 39 of the output specifications) cannot be used with UDATE, UMONTH, UDAY, and UYEAR.

PAGE, PAGE1-PAGE7

PAGE is used to number the pages of a report, to serially number the output records in a file, or to sequentially number output fields. It does not cause a page eject. PAGE1 through PAGE7 are used to serially number several output files.

The eight possible PAGE entries (PAGE, PAGE1, PAGE2, PAGE3, PAGE4, PAGE5, PAGE6, and PAGE7) may be needed for numbering different types of output pages or for numbering pages for different printer files.

PAGE and PAGE1 through PAGE7 can be specified in positions 32 through 37 of the output specifications or in the input or calculation specifications.

Rules for PAGE, PAGE1-PAGE7

Remember the following rules when using the PAGE fields:

- When a PAGE field is specified in the output specifications, without being defined elsewhere, it is assumed to be a four-digit, numeric field with zero decimal positions.
- Page numbering, unless otherwise specified, starts with 0001; and 1 is automatically added for each new page.
- To start at a page number other than 1, enter that page number in a field of an input record and name that field PAGE in positions 53 through 58, or use a calculation operation such as Z-ADD The number entered in the PAGE field should be one less than the starting page number. For example, if numbering starts with 24, enter a 23 in the PAGE field. The PAGE field can be of any length but must have zero decimal positions (see Figure 10-1 on page 10-4). Any entry in the PAGE field should be right-adjusted, such as 0023.
- Page numbering can be restarted at any point in a job. The following methods can be used to reset the PAGE field:
  - Specify blank-after (position 39 of the output specifications).
  - Specify the PAGE field as the result field of an operation in the calculation specifications.
  - Specify output indicators in the output specifications (see Figure 10-2). Output indicators cannot be used to control the printing of a PAGE field, because a PAGE field is always written.
  - Specify the PAGE field as an input field as shown in Figure 10-1.
- Leading zeros are automatically suppressed (Z edit code is assumed) when a PAGE field is printed unless an edit code, edit word, or data format (P/B/L/R in position 44) has been specified. Editing and the data format override the suppression of leading zeros.
- PAGE can be specified in input or calculation specifications, and can be of any length. When the PAGE field is defined in input and calculation specifications, it is treated as a field name in the output specifications and zero suppression is not automatic.

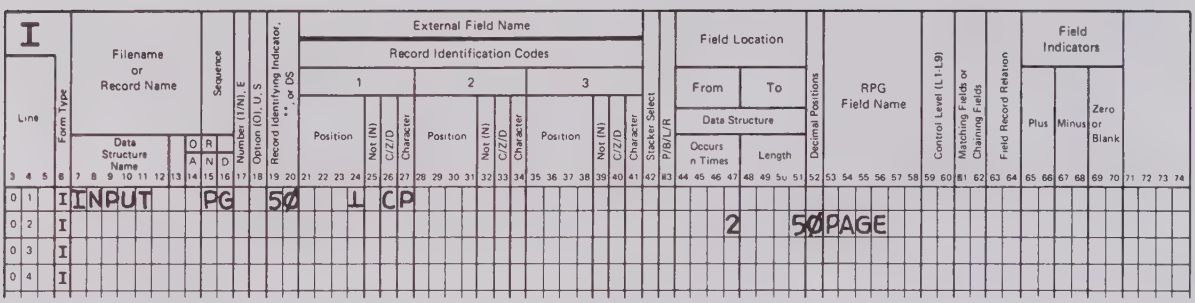
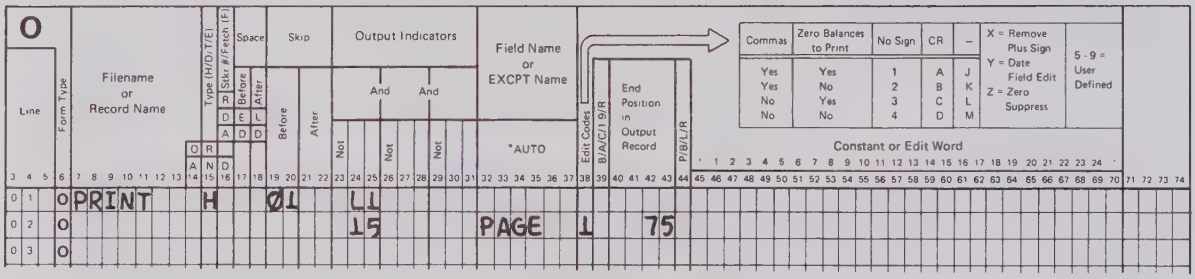


Figure 10-1. Page Record Description



When indicator 15 is on, the PAGE field is reset to zero and 1 is added before the field is printed. When indicator 15 is off, 1 is added to the contents of the PAGE field before it is printed.

Figure 10-2. Resetting the PAGE Fields to Zero

Figurative Constants (*BLANK/*BLANKS, *ZERO/*ZEROS, *HIVAL, *LOVAL, *ALL'X..')

The figurative constants *BLANK/*BLANKS, *ZERO/*ZEROS, *HIVAL, *LOVAL, and *ALL'X..' are implied literals that can be specified without a length, because the implied length and decimal positions of a figurative constant are the same as those of the associated field. (For exceptions, see the following section, "Rules for Figurative Constants" on page 10-5.)

Figurative constants can be specified in positions 18 through 27 (factor 1) and in positions 33 through 42 (factor 2) of the calculation specifications. The following shows the reserved words and implied values for figurative constants:

Reserved Words for Figurative Constants	Implied Values
*BLANK/*BLANKS	All blanks. Valid only for character fields.
*ZERO/*ZEROS	Character/numeric fields: All zeros.

Reserved Words for Figurative Constants	Implied Values
*HIVAL	Character fields: The highest collating character for the system (hexadecimal FFs). Numeric fields: All nines with a positive sign.
*LOVAL	Character fields: The lowest collating character for the system (hexadecimal zeros). Numeric fields: All nines with a negative sign.
*ALL'X..'	Character/numeric fields: Character string X . . is cyclically repeated to a length equal to the associated field. If the field is a numeric field, all characters within the string must be numeric (0 through 9). No sign or decimal point can be specified when *ALL'X . .' is used as a numeric constant.

Rules for Figurative Constants

Remember the following rules when using figurative constants:

- Figurative constants are considered elementary items. Except for MOVEA, figurative constants act like a field if used in conjunction with an array. For example:
  MOVE *ALL'XYZ'ARR.

  If ARR has 4-byte character elements, then each element will contain 'XYZX'.
- MOVEA is considered to be a special case. The constant is generated with a length equal to the portion of the array specified. For example:
  - MOVEA *BLANK ARR,X

    Beginning with element X, the remainder of ARR will contain blanks.
  - MOVEA *ALL'XYZ' ARR,X

    ARR has 4-byte character elements. Element boundaries are ignored, as is always the case with character MOVEA operations. Beginning with element X, the remainder of the array will contain 'XYZXYZXYZ...'.

Note that the results of MOVEA are different from those of the MOVE example above.

- After figurative constants are set/reset to their appropriate length, their normal collating sequence can be altered if an alternate collating sequence is specified.
- The move operations MOVE and MOVEL produce the same result when moving the figurative constant *ALL'X..'. The character string is cyclically repeated character by character (starting on the left) until the length of the associated field is the same as the length of the character string.
- Figurative constants are valid in compare operations such as COMP, CAB, DOU, DOW, IF; the associated field in the compare operations is the field with which the figurative constant is to be compared.

- Figurative constants are not allowed in factor 1 of a DEBUG or DSPLY operation or in factor 2 of MHLZO, MLHZO MHHZO, MLLZO, BITON, BITOF, TESTB, or SQRT operations.

Index

A

accumulating totals (auto report) 8-16
ACQ (acquire) 9-17
acquire (ACQ) 9-17
add a record
  file description specifications entry (A) 2-17
  output specification entry (ADD) 7-10
ADD operation code 9-18
alphameric
  field
    length of 5-17, 5-27
    moving (MOVE operations) 9-8
    resetting to blanks 7-17
  key 2-13
  literal
    considerations for 6-8
alternate collating sequence
  control specification entry 1-7
alternating format (arrays and tables)
  specification of 3-9
ampersand (&)
  in date edit 1-6
AN and OR lines
  on calculation specifications 6-7
AND and OR lines
  calculation specifications 6-7
  input specifications 5-16
  output specifications 7-9, 7-20
  *AUTO output specifications 8-12
  *AUTO page headings specifications 8-7
AND relationship
  calculation specifications 6-7
  input specifications 5-16
  output specifications 7-9, 7-20
    conditioning indicators 7-13
  *AUTO output specifications 8-12
ANDxx (And) operation code 9-19
apostrophe
  use with output constant 7-19
arithmetic operation codes
  See also calculation operations
  See also factor 1
  See also factor 2
  See also half-adjust
  See also operation codes
  ADD 9-18
  DIV (divide) 9-47
  general information 9-5
  MULT (multiply) 9-99
  MVR (move remainder) 9-100
arithmetic operation codes *(continued)*
  SQRT (square root) 9-137
  SUB (subtract) 9-138
  XFOOT (summing the elements of an array) 9-151
  Z-ADD (zero and add) 9-152
  Z-SUB (zero and subtract) 9-153
arithmetic operations 9-5
array
  alternating
    specification of 3-9
  auto report specifications
    format of compile-time arrays 8-1
    order in generated program 8-3
  building
    See array, loading
  building via calculations
    See run-time array
  combined array file 2-8
  comments 3-9
  decimal positions 3-8
  dynamic arrays
    See run-time array
  end position 7-17
  entries
    length of 3-7
    number of per array 3-7
    number per record 3-7
  extension specifications
    possible entries 3-4
    summary 3-2
  file
    description of 2-9
    sequence 3-9
  file description specifications entry 2-9
  file name (when required on file description specifications) 2-7
  format of 3-8
  from filename 3-4
  length of entry 3-8
  loading
  LOKUP operation code 9-82
  maximum number of 3-1
  moving (MOVEA operation code) 9-91
  name
    as field name in *AUTO output 8-13, 8-21
    as field name in *AUTO page headings 8-9
    extension specifications 3-4
    in compare operations 9-10
    input specifications 5-18
    output specifications 7-15

array *(continued)*
number of entries per array 3-7
number of entries per record 3-7
run-time
sequence of data 3-9
square root (SQRT) operation code 9-137
summing elements of (XFOOT) operation code 9-151
to filename 3-5
XFOOT operation code 9-151
*AUTO output field name 8-13, 8-21
instruction with A in position 39 8-16
*AUTO page headings 8-7
ascending sequence
extension specifications entry 3-9
file description specifications entry 2-10
asterisk (*)
generated specifications 8-1
generated total fields restrictions in naming fields 8-16
auto report copy function
See /COPY statement specifications
auto report function
See also auto report option specifications
See also *AUTO output specifications
See also *AUTO page headings specifications
See also /COPY statement specifications
generated specifications
format of 8-1
order of 8-2
operations of 8-2
auto report option specifications 8-4
auto report output function
See *AUTO page headings specifications
auto report page heading function
See *AUTO page headings specifications
auto report program

B

begin subroutine (BEGSR) operation code 9-20
BEGSR (begin subroutine) operation code 9-20
binary field
data structure subfield specifications 5-26
for array/table file 3-8
input specifications 5-17
output specifications 7-18
binary format
array/table field 3-8
input field 5-17
output field 7-18
binary operators 9-21, 9-22
binary relative record number 2-14
bit testing (TESTB) 9-140
BITOF (set bits off) operation code 9-21
BITON (set bits on) operation code 9-22
blank after
output specifications 7-17, 7-22
*AUTO output 8-14
generated for total fields 8-16
*AUTO page headings 8-9
branch to (GOTO) operation code 9-73
branching within RPG/400 operations 9-13

C

CABxx (compare and branch) operation code 9-24
calculation
generated by auto report 8-3
indicators
AND/OR relationship 6-7
conditioning 6-1
control level 6-6
resulting 6-11
operations 6-9
See also operation codes
summary of 9-1
specifications
entries for factor 1 6-8
entries for factor 2 6-8
entries for result field 6-9
general description 6-1
order in auto report generated program 8-3
other uses for 6-12
relationship between positions 7 and 8 and 9-17 6-7
summary of 6-3
subroutines
BEGSR (begin subroutine) operation code 9-20
coding of 9-67
ENDSR (end subroutine) operation code 9-60
EXSR (execute subroutine) operation code 9-67
SR identifier 6-7
calculation-time output (EXCPT) operation code 9-61
call a program (CALL) operation code
call and branching operation codes 9-13
CALL operation code
layout 9-27
specifications 9-27
case (CASxx) operation code 9-29
CASxx (case) operation code 9-29
CHAIN operation code 9-31
character
in record identification code 5-16
checking sequence
See sequence checking
CL programs
close files (CLOSE) operation code 9-34

CLOSE operation code 9-34
code
See edit codes
codes, operation
See operation codes
coding form
See calculation specifications
See control specifications
See data description specifications
See extension specifications
See file description specifications
See input specifications
See line counter specifications
See output specifications
See translation table and alternate collating sequence coding sheet
coding subroutines 9-67
collating sequence
See alternate collating sequence
column headings
See *AUTO page headings specifications
combined file
description 2-8
COMIT (Commit) 2-21, 9-36
comments
on calculation specifications 6-12
on extension specifications 3-10
order in auto report generated program 8-4
Commit (COMIT) 9-36
COMP operation code 9-37
compare and branch (CABxx) operation code 9-24
compare operations 9-9
compare (COMP) operation code 9-37
compile-time array or table
See array
composite key operation codes
KFLD (define parts of a key) 9-80
KLIST (define a composite key) 9-80
conditioning calculations 6-1
conditioning files 2-18
See also external indicators
conditioning indicators
calculation
specification of 6-8
file
specification of 2-18
output
controlling a record 7-13
controlling fields of a record 7-15
specification of 7-13
conditioning output
for fields of a record 7-15
for records 7-13
constants
See also edit word
constants *(continued)*
See also literals
figurative 10-4
rules for use on output specification 7-19
*AUTO output
A in position 39 8-15
blank or B in position 39 8-13
C in position 39 8-19
1-9 or R in position 39 8-20
*AUTO page headings 8-7
constants, figurative
See *ALL'X..',
See *BLANK/*BLANKS
See *HIVAL
See *LOVAL
See *ZERO/*ZEROS
continuation line 2-6, 2-16, 2-18
continuation line option
INFDS (file information data structure) 2-22
INFSR (file exception/error subroutine) 2-22
PLIST (parameter list for SPECIAL files) 2-23
PRTCTL (printer control) 2-23
RECNO (relative record number field) 2-24
SFILE (WORKSTN subfile record) 2-25
SLN (Start Line Number) 2-25
continuation record
See continuation line
control break
See control field
See control group
See control level (L1-L9) indicators
control field
See also control break
See also control level (L1-L9) indicators
assigning on input specifications
externally described file 5-23
program described file 5-18
control group
See control break
See control field
See control level (L1-L9) indicators
control language (CL) program
control level (L1-L9) indicators
See also conditioning calculations
See also control break
See also control field
See also control group
as field record relation indicator 5-20
as record identifying indicator 5-13, 5-21
assigning to input fields 5-18, 5-23
conditioning calculations 6-1
conditioning output 7-13
with auto report 8-16, 8-21

control specification
data area (RPGHSPEC) 1-1
general description 1-1
summary of 1-3
copy function, auto report
See /COPY statement specifications
CR (negative balance symbol)
C/Z/D (character/zone/digit)
in record identification code for program described file 5-15

D

D-*AUTO
See *AUTO output specifications
data area data structure
renaming 5-24
statement
externally described 5-25
program described 5-25
specifications 5-24
subfields
specifications 5-26
data area operations
general information 9-15
IN (retrieve a data area) 9-77
OUT (write a data area) 9-107
UNLCK (unlock a data area) 9-146
data areas
defining 9-43
local 9-43
restrictions 9-43
retrieval
explicit 9-77
unlocking
explicit 9-105
writing
explicit 9-107
data base
file
See ?
data format
See binary format
See packed decimal format
data management feedback area
See file information data structure
data structure
length of 5-26
multiple occurrences of 5-26
name 5-24
on input specifications 5-24
renaming 5-25
subfields
external name 5-26
format 5-26
location 5-27
data structure *(continued)*
subfields *(continued)*
RPG/400 name 5-27
specifications 5-26
with OCUR operation code 9-102
date edit 1-6
date field
effect of inverted print 1-6
relation to entry in position 21 of control specification 1-6
date format 1-6
date generated for *AUTO page headings 8-7
date suppression (auto report) 8-5
date, user
See UDATE
See UDAY
See UMONTH
See UYEAR
deactivate a program (FREE) operation code 9-71
debug function (DEBUG) operation code 9-39
DEBUG operation code
control specification for 1-5
description 9-39
records written 9-40
debugging RPG programs
See breakpoint trace
See DEBUG and DUMP operation codes
decimal data format
See packed decimal format
decimal positions
calculation specifications 6-10
extension specifications 3-8
generated total fields (auto report) 8-16
input specifications
data structure subfield entry 5-27
field description entry for program described file 5-18
with arithmetic operations 9-5
declarative operation codes 9-10
define a composite key (KLIST) operation code 9-80
define parts of a key (KFLD) operation code 9-79
DEFN (define operation code) 9-42
delete a record
DELET (delete record) operation code 9-46
output specifications entry (DEL) 7-10
descending sequence
extension specifications entry 3-9
file description specifications entry 2-10
detail lines
See *AUTO output specifications
detail printing, auto report
See *AUTO output specifications
detail (D) output record 7-10
display device file
See WORKSTN file

display function (DSPLY) operation code 9-54
distributed data management (DDM) xv
DIV (divide) operation code 9-47
  See also MVR operation code
divide (DIV) operation code 9-47
do group
  general description 9-11
DO operation code 9-48
do until (DOUxx) operation code 9-50
do while (DOWxx) operation code 9-52
double asterisk (**)
  generated specifications 8-1
  generated total fields restrictions in naming fields 8-16
DOUxx (do until) operation code 9-50
DOWxx (do while) operation code 9-52
DS
  See also data structure
  input specifications entry 5-24
DSPLY (display function) operation code 9-54
DUMP (program dump) operation code
  control specification for 1-5
  general information 9-57
dynamic array
  See run-time array

E

edit codes
  date field
    relation to entry in position 21 in control specification 1-6
  *AUTO output specification 8-14
    generated K edit code 8-14, 8-18
  *AUTO page headings specifications 8-9
edit word
  on output specifications 7-19
editing
edit, date 1-6
ELSE (else do) operation code 9-58
End Job (ENDJOB) 9-115
end of file
  file description specifications entry 2-10
END operation code 9-59
end position
  in output record
    for RPG output specifications 7-17
    for *AUTO output specifications 8-14
end subroutine (ENDSR) operation code 9-60
End Subsystem (ENDSBS) 9-115
End System (ENDSYS) 9-115
end (END) operation code 9-59
ENDCPF (End System) 9-115
ENDJOB (End Job) 9-115
ENDSBS (End Subsystem) 9-115
ENDSR (end subroutine) operation code
  description 9-60
exception (E) output records 7-10
  See also EXCPT operation code
EXCPT name
  on output specifications 7-14
EXCPT (calculation time output) operation code 9-61
EXFMT (execute format) operation code 9-66
  description 9-66
EXSR (run subroutine) operation code 9-67
extension code (file description specifications entry for program described file) 2-15
extension specifications
  entry on file description specifications 2-15
  file entries 3-4
  general description 3-1
  possible entries 3-4
  summary of 3-2
external field name
  renaming 5-22
external message queue (*EXT) 9-55
external (U1-U8) indicators
  as field indicator 5-20, 5-24
  as field record relation indicator 5-20
  as record identifying indicator 5-12, 5-22
  assigning on file description specifications 2-18
  conditioning calculations 6-8
  conditioning output 7-13
  resetting 5-20
  used to condition files 2-18
externally described file
  file description specifications for 2-1
  input specifications for 5-21
  names 2-7
  output specifications for 7-19

F

factor 1
  as search argument 9-82
  entries for 6-8
  in arithmetic operations 9-5
  with CABxx operation 9-24
  with COMP operation 9-37
factor 2
  entries for 6-9
  in arithmetic operations 9-5
  with CABxx operation 9-24
  with COMP operation 9-37
FEOD (force end of date) operation code 9-69
fetch overflow
  See also overflow (OA-OG, OV) indicators
  entry on output specifications 7-11
  general description 7-11

fetch overflow *(continued)*
relationship with 7-14
relationship with OR line 7-14
field
alphameric
with program described file on input specification 5-17
with subfield specifications on input specification 5-27
binary
on extension specifications 3-8
on input specifications 5-17, 5-27
on output specifications 7-18
defining as data area 9-43
defining new 6-10
key 2-12, 2-15
length 5-17
lookahead
with program described file 5-13
numeric
on output specifications 7-15
with data structure subfield specifications on input specifications 5-27
with program described file on input specifications 5-17
record address 2-12
result 6-9
zeroing 7-17
field definition (DEFN) operation code 9-42
field indicators (01-99, H1-H9, U1-U8, RT)
assigning on input specifications
for externally described files 5-23
for program described files 5-20
conditioning calculations 6-7
conditioning output 7-13
field length
alphameric 5-17, 5-26
arithmetic operations 9-5
calculation operations 6-10
calculation specifications 6-10
compare operations 9-9
input specifications 5-17, 5-27
key 2-12
numeric 5-17, 5-27
record address 2-12
field location entry (input specifications)
for program described file 5-17
field name
as result field 6-9
external 5-23
generated by auto report 8-16
in an OR relationship 5-16
on output specifications 7-15
special words as 7-15, 10-2
field record relation indicators (01-99, H1-H9, L1-L9, U1-U8)
assigning on input specifications 5-19
figurative constants 10-4
See also *ALL'X..'
See also *BLANK/*BLANKS
See also *HIVAL
See also *LOVAL
See also *ZERO/*ZEROS
rules for 10-5
file
adding records to 2-16, 7-10
addition 2-16
array
See array
combined 2-8
conditioning 2-17
data base
See ?
deleting existing records from 7-10
deleting records from
DEL 7-10
DELET 9-46
DISK
See ?
display device
See WORKSTN file
end of 2-9
externally described
See externally described file
format 2-11
full procedural 2-9
indexed 2-14
input 2-8
name
entry on extension specifications 3-4, 3-5
entry on file description specifications 2-6
entry on input specifications 5-11
entry on line counter specifications 4-3
entry on output specifications 7-8
externally described 2-7
program described 2-7
nonkeyed program described 2-14
number allowed on file description specifications 2-1
opening, user control of 2-18
organization 2-14
output 2-8
primary 2-9
PRINTER
See PRINTER file
program described
See program described file
record address
See record address file

file *(continued)*
secondary 2-9
SEQ
See SEQ file
SPECIAL
See SPECIAL file
table
See table
tape
See SEQ file
types 2-8
update 2-8
WORKSTN
See WORKSTN file
file condition 2-17
See also external indicators, UC
file description specifications
general description 2-1
summary of 2-3
file designation 2-8
See also array or table file
See also full procedural file
See also output file
See also primary file
See also record address file
See also secondary file
file exception/error subroutine (INFSR)
continuation line option 2-22
file format 2-11
file information data structure
continuation line option 2-21
entry on file description specifications 2-21
statement
specifications 5-14
subfields
specifications 5-17
file operation codes
CHAIN (random retrieval from a file based on record number or key value) 9-31
CLOSE (close files) 9-34
DELET (delete record) 9-46
EXFMT (execute format) 9-66
FEOD (force end of data) 9-69
general description 9-13
OPEN (open file for processing) 9-105
READ (read a record) 9-114
READC (read next modified record) 9-117
READE (read equal key) 9-118
READP (read prior record) 9-120
SETGT (set greater than) 9-125
SETLL (set lower limit) 9-129
UPDAT (modify existing record) 9-147
WRITE (create new records) 9-149
first page (1P) forms position 1-7
first page (1P) indicator
conditioning output 7-13, 7-16
floating dollar sign, auto report 8-20
force a certain file to be read on the next cycle (FORCE) operation code 9-70
force end of data (FEOD) operation code 9-69
FORCE operation code 9-70
form length
See also line counter specifications
default value 4-1
how to specify 4-3
overrides 4-1
format, date 1-6
forms position, 1P 1-7
FREE (deactivate a program) operation code specifications 9-71
from filename (extension specifications) 3-4
full procedural file
description of 2-9
file description specifications entry 2-8
file operation codes 9-13

G

generated RPG/400 program, auto report
calculations 8-3
comments 8-3
date 8-7
date suppression 8-7
edit codes 8-14, 8-18
field names 8-13
format of generated specifications 8-1
order of specifications 8-2
output specifications 8-3
page number 8-7
total fields 8-16
generated specifications, auto report 8-1
generated total fields, auto report 8-16
get-device-attribute
See POST (Post)
GOTO (branch to) operation code 9-73
See also TAG operation code

H

H-*AUTO record description 8-7
See also *AUTO page headings specifications
half-adjust
on calculation specifications 6-11
operations allowed with 6-11
halt (H1-H9) indicators
as field indicators 5-20, 5-24
as field record relation indicator 5-20
as record identifying indicator 5-12, 5-22
as resulting indicator 6-11

halt (H1-H9) indicators *(continued)*
conditioning calculations 6-8
conditioning output 7-13, 7-15
header specifications
See control specification
heading (H) output records 7-9
headings
See column headings
See *AUTO page headings specifications
H1-H9
See halt (H1-H9) indicators

I

ID entry 2-21
IFxx (IF/THEN) operation code 9-75
IF/THEN (IF) operation code 9-75
implied literals
See figurative constants
IN (retrieve a data area) 9-77
IND entry 2-22
indexed file
format of keys 2-14
key field 2-15
processing 2-14
indicators
See also individual operation codes
calculation specifications 6-11
control level (L1-L9)
as field record relation indicator 5-18
as record identifying indicator 5-13, 5-23
assigning to input fields 5-18, 5-23
conditioning calculations 6-7
conditioning output 7-13, 7-15
with auto report 8-16, 8-21
external (U1-U8)
as field record relation indicator 5-20
assigning on file description specifications 2-17
conditioning calculations 6-8
conditioning output 7-13
resetting 5-20
used to condition files 2-17
field
assigning on input specifications 5-20, 5-23
conditioning calculations 6-8
conditioning output 7-13
field record relation
assigning on input specifications 5-19
first page (1P)
conditioning output 7-13, 7-16
halt (H1-H9)
as field record relation indicator 5-20
as resulting indicator 6-11
conditioning calculations 6-7
conditioning output 7-13, 7-15
indicators *(continued)*
last record (LR)
as record identifying indicator 5-12, 5-22
as resulting indicator 6-11
conditioning calculations 6-7, 6-8
conditioning output 7-13, 7-15
level zero (L0)
calculation specifications 6-6
matching record (MR)
See also multifile processing
as field record relation indicator 5-20
conditioning calculations 6-7
conditioning output 7-13, 7-15
output
AND/OR lines 7-15
assigning 7-13
restriction in use of negative indicators 7-13
overflow
assigning on file description specifications 2-15
conditioning auto report page headings 8-7
conditioning calculations 6-8
conditioning output 7-13, 7-15
with exception lines 7-14, 9-61
record identifying
conditioning calculations 6-8
conditioning output 7-13, 7-15
return (RT)
as record identifying indicator 5-13, 5-22
as resulting indicator 6-11
conditioning calculations 6-8
INFDS
See file information data structure
information operation codes
DEBUG (debug function) 9-39
DUMP (program dump) 9-57
general information 9-14
SHTDN (shut down) 9-135
TIME (time of day) 9-145
INFSR
See file exception/error subroutine (INFSR)
input fields
as look-ahead field 5-13
decimal positions 5-18
external name 5-21
format of 5-17
location 5-17
name of 5-18
RPG/400 name of 5-23
input file 2-8
input specification indicators
control level 5-18, 5-23
field record relation indicator 5-19
input specifications
for data structures 5-24
for externally described file 5-21

input specifications *(continued)*
for program described file 5-10
general description 5-1
summary of 5-3
input specifications for externally described file
control level indicators 5-23
external field name 5-22
field indicators 5-23
match fields 5-23
record identifying indicator 5-22
record name 5-21
RPG/400 field name 5-23
input specifications for program described file
AND relationship 5-16
field
decimal positions 5-18
format 5-17
location 5-17
name 5-18
filename 5-11
indicators
control level 5-18
field 5-17
field record relation 5-20
record identifying 5-13
look-ahead field 5-13
match fields 5-19
number of records 5-12
option 5-12
record identification codes 5-14
sequence checking 5-11
inverted print 1-6
INVITE DDS keyword 9-115

K

key field
alphanumeric 2-12
for externally described file 2-12
format of 2-13
length of 2-12
location of 2-15
packed 2-12
keyed processing
indexed file 2-14
specification of keys 2-12
KFLD (define parts of a key) operation code 9-79
KLIST (define a composite key) operation code 9-80
general information 9-80

L

last record (LR) indicator
as record identifying indicator 5-12, 5-22
as resulting indicator 6-11
last record (LR) indicator *(continued)*
conditioning calculations
positions 7 and 8 6-6, 6-7
positions 9-17 6-7
conditioning output 7-13, 7-15
level zero (L0) indicator
calculation specifications 6-6
line counter specifications 4-1
summary of 4-2
lines per page 4-3
literals
alphameric 6-8
implied
See figurative constants
numeric 6-9
local data area 9-43
logical file
See ?
LOKUP (look up) operation code 9-82
look up (LOKUP) operation code 9-82
look-ahead field 5-13
loop
LR (last record) indicator
See last record (LR) indicator
L0 indicator
See level zero (L0) indicator
L1-L9 (control level) indicators
See control level (L1-L9) indicators

M

match fields
See also multifile processing
input specifications for 5-19, 5-23
matching record (MR) indicator
See also multifile processing
as field record relation indicator 5-20
assigning match fields 5-19, 5-23
conditioning calculations
positions 7 and 8 6-6
positions 9-17 6-8
conditioning output 7-13, 7-15
maximum number of files allowed 2-1
number allowed on file description specifications 2-1
message identification 9-54
message operation code
DSPLY (display function) 9-54
MHHZO (move high to high zone) operation code 9-85
MHLZO (move high to low zone) operation code 9-86
mixed file
See WORKSTN file
MLHZO (move low to high zone) operation code 9-87

MLLZO (move low to low zone) operation code 9-88
move array (MOVEA) operation code 9-91
move high to high zone (MHHZO) operation code 9-85
move high to low zone (MHLZO) operation code 9-86
move left (MOVEL) operation code 9-96
move low to high zone (MLHZO) operation code 9-87
move low to low zone (MLLZO) operation code 9-88
MOVE operation code 9-89
move operation codes
  MOVE 9-89
  MOVEA (move an array) 9-91
  MOVEL (move left) 9-96
move operations 9-8
move remainder (MVR) operation code 9-100
move zone operation codes
  MHHZO (move high to high zone) 9-85
  MHLZO (move high to low zone) 9-86
  MLHZO (move low to high zone) 9-87
  MLLZO (move low to low zone) 9-88
move zone operations 9-9
MOVEA (move array) operation code 9-91
MOVEL (move left) operation code 9-96
MR (matching record) indicator
  See matching record (MR) indicator
MULT (multiply) operation code 9-99
multifile processing
  FORCE operation code 9-70
multiply (MULT) operation code 9-99
MVR (move remainder) operation code 9-100

N

name(s)
  EXCPT 7-14
  field
    on input specifications 5-18, 5-23
    on output specifications 7-13
  for *ROUTINE
negative balance (CR)
nested do group
  example 9-12
NEXT (Next) 9-101
nonkeyed processing 2-13
NUM entry 2-23
numeric fields
  auto report
    editing 8-18
  resetting to zeros 7-17
numeric literal
  considerations for use 6-9
numeric (01-99) indicators
  See field and field record relation indicators
  See indicator conditioning calculations and output
  See record identifying and resulting indicators

O

OA-OG, OV (overflow) indicators
  See overflow (OA-OG, OV) indicators
OCUR (set/get occurrence of a data structure (operation code)) 9-102
OPEN operation code 9-105
  specifications for 9-105
operation codes
  arithmetic
    ADD 9-18
    DIV (divide) 9-47
    general information 9-5
    MULT (multiply) 9-99
    MVR (move remainder) 9-100
    SQRT (square root) 9-137
    SUB (subtract) 9-138
    XFOOT (summing the elements of an array) 9-151
    Z-ADD (zero and add) 9-152
    Z-SUB (zero and subtract) 9-153
  bit
    BITOF (set bits off) 9-21
    BITON (set bits on) 9-22
  call and branching
    CALL (call a program) 9-27
    EXCEPT (calculation time output) 9-61
    FREE (deactivate a program) 9-71
    general information 9-13
    GOTO (transfer control to) 9-73
    RETURN (return to caller) 9-123
  compare
    CABxx (compare and branch) 9-24
    COMP (compare) 9-37
    general information 9-9
  data area
    general information 9-15
    IN (retrieve a data area) 9-77
    OUT (write a data area) 9-107
    UNLCK (unlock a data area) 9-146
  declarative
    DEFN (field definition) 9-42
  file
  force a certain file to be read on next cycle (FORCE) 9-70
  general information 9-10
    KFLD (define parts of a key) 9-79
    KLIST (define a composite key) 9-80
    PARM (identify parameters) 9-109
    PLIST (identify a parameter list) 9-111
    TAG 9-139
  information
    DEBUG (debug function) 9-39
    DUMP (program dump) 9-57
    general information 9-14
    SHTDN (shut down) 9-135

operation codes *(continued)*
information *(continued)*
TIME (time of day) 9-145
lookup (LOKUP) 9-82
message
DSPLY (display function) 9-54
move
general information 9-8
MOVE 9-89
MOVEA (move an array) 9-91
MOVEL (move left) 9-96
move zone
general information 9-9
MHHZO (move high to high zone) 9-85
MHLZO (move high to low zone) 9-86
MLHZO (move low to high zone) 9-87
MLLZO (move low to low zone) 9-88
setting indicators
general information 9-11
SETOF (set off) 9-133
SETON (set on) 9-134
set/get occurrence of a data structure (OCUR) 9-102
sort an array (SORTA) 9-136
structured programming
ANDxx 9-19
DO 9-48
DOUxx (do until) 9-50
DOWxx (do while) 9-52
ELSE (else do) 9-58
END 9-59
general information 9-11
IFxx (IF/THEN) 9-75
ORxx 9-106
subroutine
BEGSR (begin subroutine) 9-20
ENDSR (end subroutine) 9-60
EXSR (execute subroutine) 9-67
general information 9-10
summary of operation codes 9-1
testing
TESTB (test bit) 9-140
TESTN (test numeric) 9-142
TESTZ (test zone) 9-144
OR lines
on calculations 6-8
on input specifications 5-16
on output specifications 7-9, 7-20
ORxx (or) operation code 9-106
OUT (write a data area) 9-107
output
conditioning indicators 7-13
file 2-8
indicators
See also *AUTO output specifications
See also *AUTO page headings specifications
output *(continued)*
indicators *(continued)*
AND/OR lines 7-13, 7-15
assigning 7-13
restrictions in use of negative indicators 7-13
record
end position in 7-17
specifications
for externally described file 7-19
for fields of a record 7-15
for program described file 7-8
for records 7-9
general description 7-1
generated by auto report 8-2
output file 2-8
output specifications
for externally described files 7-19
for program described files 7-8
summary of 7-3
output specifications for externally
indicators 7-20
record name 7-20
record type 7-20
output specifications for externally described file
ADD records 7-20
AND/OR lines 7-20
DEL (delete) records 7-20
EXCPT name 7-21
field name 7-21
*ALL 7-21
output specifications for program described file
ADD record 7-10
AND/OR lines 7-9
blank after 7-17
conditioning indicators 7-13
DEL (delete) record 7-10
detail record 7-10
edit codes 7-16
end position of field 7-17
exception record 7-10
EXCPT name 7-14
field
format of 7-19
name 7-15
file name 7-9
PAGE, PAGE 1-PAGE7 7-15
UDATE 7-15
UDAY 7-15
UMONTH 7-15
UYEAR 7-15
*IN, *INxx, *IN ,xx 7-16
*PLACE 7-16
overflow
line number 4-3
lines (number of) 4-3

overflow indicators
assigning on file description specifications 2-14
conditioning auto report page headings 8-7
conditioning calculations 6-7
conditioning output 7-13
with exception lines 7-14, 9-58
overview of auto report 8-1

P

packed decimal format
keys 2-13
page numbering
See PAGE, PAGE1-PAGE7
PAGE, PAGE1-PAGE 7 7-16
PAGE, PAGE1-PAGE7 10-3
parameter list
See PARM (identify parameters) operation code
PARM (identify parameters) operation code
calculation specifications 9-109
PASS keyword 2-23
PLIST keyword for SPECIAL file
continuation line option 2-23
PLIST (identify a parameter list) operation code
calculation specifications 9-111
*ENTRY PLIST 9-111
POST (Post) 9-113
Power Down System (PWRDWNSYS) 9-115
prerun-time array or table
See array
primary file
file description specifications 2-9
general description 2-9
printer control option
See PRTCTL
PRINTER file
device name 2-15
form length 4-3
number of lines per page 4-3
overrides for form length 4-1
processing
methods
keyed (see keyed processing)
program described file
entries on
file description specifications 2-1
input specifications 5-1, 5-10
output specifications 7-1
length of key field 2-12
length of logical record 2-11
program dump (DUMP) operation code 9-57
program identification
See program name
program name
default 1-8
program name *(continued)*
on control specification 1-8
program status data structure
subfields
with OCUR operation code 9-102
protecting records/files
See file locking by RPG
PRTCTL (printer control)
continuation line option 2-23
relationship to positions 60-65 on file description specifications 2-23
with space/skip entries 7-11
PWRDWNSYS (Power Down System) 9-115

Q

QSYSOPR 9-55
queues
QSYSOPR 9-55
*EXT (external message) 9-55

R

random retrieval from a file based on record number or key value (CHAIN)
operation code 9-31
READ (read a record) operation code
specifications for 9-114
READC (read next modified record) operation code
specifications for 9-117
READE (read equal key) operation code 9-118
READP (read prior record) operation code 9-120
RECNO
continuation line option 2-24
record
adding to a file 2-17, 7-10
deleting from a file 7-10, 9-46
detail (D) 7-10
exception (E) 7-10
with EXCPT operation code 9-61
externally described 7-19
heading (H) 7-9
input specifications
externally described file 5-21
program described file 5-10
length 2-11
output specifications
externally described 7-19
program described 7-9
renaming 2-24
total (T) 7-10
record address file
description 2-9
extension specifications entry 3-4
file description specifications entry 2-8

record address file *(continued)*
format of keys 2-12
length of record address field 2-12
number allowed per program 2-9
relative record number 2-14
restrictions 2-9
sequential-within-limits 2-11
record address limits file
See record address file
record address relative record number file
See record address file
record identification codes 5-14
record identification entries
input specifications 5-10, 5-21
output specifications 7-8, 7-19
*AUTO output specifications 8-10
*AUTO page headings specifications 8-7
record identifying indicators (01-99, H1-H9, L1-L9, LR, U1-U8, RT)
assigning on input specifications
for externally described file 5-21
for program described file 5-10
conditioning calculations 6-6, 6-7
conditioning output 7-13, 7-15
record name
for externally described input file 5-21
for externally described output file 7-20
record sharing
See file locking by RPG
REL (release) 9-122
relative record number record address file
See record address file
Release (output specifications) 7-11, 7-20
release (REL) 9-122
requester
accessing with ID 2-21
reserved words
PAGE 7-16, 10-3
PAGE1-PAGE7 7-16, 10-3
UDATE 1-6, 10-2
UDAY 10-2
UMONTH 10-2
UYEAR 10-2
*ALL 7-21
*ALL 'X..' 10-5
*BLANK/*BLANKS 10-4
*HIVAL 10-4
*LOVAL 10-4
*PLACE 7-16
*ZERO/*ZEROS 10-4
result field
length of 6-10
number of decimal positions 6-10
possible entries 6-9
resulting indicators (01-99, H1-H9, OA-OG, OV, L1-L9, LR, U1-U8, KQ-KN, KP-KY, RT)
See also individual operation codes
calculation specifications 6-11
retrieval of data area
explicit 9-77
retrieve a data area (IN) operation code 9-77
RETRN (return to caller) operation code
specifications 9-123
return (RT) indicator
as field indicator 5-20, 5-24
as record identifying indicator 5-12, 5-13, 5-22
as resulting indicator 6-11
conditioning calculations 6-8
conditioning output 7-13
ROLBK (Rollback) 9-124
Rollback (ROLBK) 9-124
RPGHSPEC data area 1-1
RPGOBJ 1-8
RT (return) indicator
See return (RT) indicator
run format (EXFMT) operation code 9-66
run subroutine (EXSR) operation code 9-67
run-time array
See array

S

SAVDS entry 2-24
secondary file
file description specifications 2-9
general description 2-9
SEQ file
sequence
ascending 2-10
collating
See alternate collating sequence
descending 2-10
sequence checking
on input specifications 5-11
with match fields 5-19
sequential within limits processing
file description specifications entry 2-11
set bits off (BITOF) operation code 9-21
set bits on (BITON) operation code 9-22
set greater than (SETGT) operation code 9-125
set lower limits (SETLL) operation code 9-129
set off (SETOF) operation code 9-133
set on and set off operations 9-11
set on (SETON) operation code 9-134
SETGT (set greater than) operation code 9-125
SETLL (set lower limits) operation code 9-129
SETOF (set off) operation code 9-133
SETON (set on) operation code 9-134

set/get occurrence of a data structure (OCUR) operation code 9-102
SFILE
See also subfiles
continuation line option 2-25
SHTDN (shut down) operation code 9-135
shut down (SHTDN) operation code 9-135
skipping
for printer output 7-11
SLN (Start Line Number) field 2-25
sort an array (SORTA) operation code 9-136
SORTA (sort an array) operation code 9-136
source program
RPG
order of specifications 1
spacing
for printer output 7-11
not with WRITE operation 9-149
SPECIAL file
special words 10-2
specification forms
overview 1
SQL statements 6-12
SQRT (square root) operation code 9-137
square root (SQRT) operation code 9-137
SR (subroutine identifier) 6-6, 6-7
structured programming operation codes
DO 9-48
DOUxx (do until) 9-50
DOWxx (do while) 9-52
ELSE (else do) 9-58
END 9-59
general information 9-11
IFxx (IF/THEN) 9-75
SUB (subtract) operation code 9-138
subfields
for data structure subfield specifications 5-26
subfields within a data structure
specifications for 5-26
subfiles
continuation line option 2-25
subroutine identifier (SR) 6-7
subroutine operation codes
BEGSR (begin subroutine) 9-20
ENDSR (end subroutine) 9-60
EXSR (execute subroutine) 9-67
general information 9-10
subroutine operations 9-10
subroutines
calculation specifications entry in positions 7 and 8 6-6, 6-7
description 9-10
example 9-67
maximum allowed per program 9-67
operation codes 9-10
subtract (SUB) operation code 9-138
summary tables
calculation specifications 6-3
control specifications 1-3
extension specifications 3-2
file description specifications 2-3
file operation codes allowed with
input specifications 5-3
line counter specifications 4-2
operation codes 9-1
output specifications 7-3
suppressing asterisks indication (auto report total line) 8-5
suppressing date and page (auto report) 8-5

T

T-*AUTO
See *AUTO output specifications
table
See also array
alternating 3-9
file 2-9
format of 3-8
from filename 3-4
maximum number of 3-1
searching
See LOKUP operation
to filename 3-5
*AUTO output field name 8-13
restriction with A in position 39 8-15
*AUTO page headings 8-7
TAG operation code 9-139
test bit (TESTB) operation code 9-140
test numeric (TESTN) operation code 9-142
test zone (TESTZ) operation code 9-144
TESTB (test bit) operation code 9-140
testing fields
See field indicators
testing RPG programs
TESTN (test numeric) operation code 9-142
TESTZ (test zone) operation code 9-144
time of day (TIME) operation code 9-145
time out 9-115
TIME (time of day) operation code 9-145
to filename (extension specifications) 3-5
total (T) output records 7-10
translation, file
See file translation
triple asterisk (***)
generated specifications 8-1
generated total fields restrictions in naming fields 8-16

U

UC 2-18
UDATE 10-2
UDAY 10-2
UMONTH 10-2
UNLCK (unlock a data area) 9-146
unlock a data area (UNLCK) operation code 9-146
UPDAT (modify existing record) operation code 9-147
  specifications for 9-147
update file 2-8
updating records/files
  See file blocking by RPG
usage of indicators
  See indicators
user control of file opening 2-18
user date 10-2
  See also UDATE
  See also UDAY
  See also UMONTH
  See also UYEAR
using auto report 8-1
UYEAR 10-2
U1-U8
  See external (U1-U8) indicators

V

variable line number 2-25

W

WAITRCD 9-115
WORKSTN file
  device name 2-15
  subfiles
WRITE (create new records) operation code
  general description 9-149

X

XFOOT (summing the elements of an array) 9-151

Y

Y edit code
  control specification entries (positions 19 through 21) 1-6

Z

Z-ADD (zero and add) operation code 9-152
Z-SUB (zero and subtract) operation code 9-153
zero (blanking) fields 7-17

Numerics

01-99 indicators
  See field and field record relation indicators
  See indicators conditioning calculations and output
  See record identifying indicators and resulting indicators
1P forms position 1-7
1P (first page) indicator
  conditioning output 7-13, 7-16

Special Characters

& (ampersand)
  in date edit 1-6
$ (fixed or floating currency symbol)
* (asterisk)
  generated specifications 8-1
  generated total fields restrictions in naming fields 8-16
  indication of auto report total line 8-18
*ALL 7-21
*ALL 'X..' 10-4
*AUTO output specifications
  See also auto report function
  accumulating totals 8-16
    resetting total fields to zero 8-18
  asterisk indication 8-18
  column headings 8-19
  detail printing 8-10
  end position entries 8-14, 8-19
  field description specifications
    A in position 39 8-15
    blank or B in position 39 8-13
    C in position 39 8-19
    1-9 or R in position 39 8-20
  field or literal on generated total line 8-20
  generated edit code 8-14
  generated RPG/400 specifications 8-2
  generated total fields 8-16
    restrictions in naming fields 8-16
  indicators, output 8-12
  number of printer files allowed 8-6
  record description entry 8-11
  spacing and skipping 8-12
  suppress asterisk indication 8-5
  total rolling 8-15
*AUTO page headings specifications 8-7
  editing 8-9
  field description entries 8-9
  generated date 8-7
  generated page number 8-7
  indicators, output 8-8
  number of heading lines allowed 8-7
  page number 8-7
    suppression of 8-7

*AUTO page headings specifications *(continued)*
record description entries 8-7
spacing and skipping 8-7
*AUTO specifications
See *AUTO output specifications
*BLANK/*BLANKS 10-4
*ENTRY PLIST 9-111
*EXT 9-55
*HIVAL 10-4
*LDA 9-43
*LIKE 9-42
*LOVAL 10-4
*M 9-54
*NAMVAR 9-43
*NOIND 2-23
*PLACE 7-16
*suppress (auto report) 8-5
*ZERO/*ZEROS 10-4
**
generated specifications 8-1
generated total fields restrictions in naming fields 8-16
look-ahead fields 5-13
***
generated specifications 8-1
generated total fields restrictions in naming fields 8-16
/COPY statement specifications
See also auto report function
format of 8-22
/END EXEC delimiter 6-12
/EXEC SQL delimiter 6-12

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your nearest IBM branch office. You may use this form to communicate your comments about this publication, its organization, or subject matter **with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.**

☐ If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your nearest IBM branch office to request additional publications.

Name _____

Company or Organization _____

Address _____

_____

SC09-1089-00

Cut or Fold Along Line

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Development Laboratory
Information Development, Department 245
Rochester, Minnesota 55901

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

International Business Machines Corporation

Program Number
5728-RG1

SC09-1089-00

Printed in U.S.A.

Technical Newsletter

This Newsletter No.	SN09-1561-00
Date	October 1988
Base Publication No.	SC09-1161-00
Prerequisite Newsletters	None

AS/400™ Languages: RPG/400 User's Guide

© Copyright International Business Machines Corporation 1988

This technical newsletter, part of Release 1.2 of AS/400 RPG/400, contains replacement pages for the *AS/400 Languages: RPG/400 User's Guide*, SC09-1161-00. Revised figures are marked in the left-hand margin of the pages with a vertical bar.

Changes include a new compiler parameter (SAAFLAG), increased discussion of the System/38 RPG, RPG/400, and System/38 Environment Option of the RPG/400 compilers, and expanded explanation of error messages.

These pages should be added to your *RPG/400 User's Guide* as described below:

Pages	Should be placed here
iv to xi	replaces iv to xi
xiii to xv	replaces xiii to xv
3-6 to 3-16	replaces 3-6 to 3-14
4-2 to 4-8b	replaces 4-2 to 4-7
G-1 to G-4	replaces G-1 to G-2
X-16	replaces X-16

IBM Canada Ltd.
Information Development,
Department 849,
1150 Eglinton Ave. East,
North York, Ontario, Canada. M3C 1H7

IBM is a registered trademark of International Business Machines Corporation, Armonk, NY.
AS/400 is a trademark of International Business Machines Corporation, Armonk, NY.

Contents

Chapter 1. An Introduction to RPG/400 and the AS/400 System 1-1
OS/400 1-1
The AS/400 Control Language 1-1
Accessing RPG/400 from the AS/400 System 1-2
Your Choice of Two Environments: AS/400 Native or the System/38 Environment 1-3
The System/38 Environment 1-3
Available Utilities and Languages 1-4
The Source Entry Utility 1-4
The Screen Design Aid 1-4
The Structured Query Language 1-4
Restrictions 1-5
RPG/400 Program Cycle 1-5
General RPG/400 Program Cycle 1-6
Designing Your RPG/400 Program 1-8
Designing the Output 1-8
Designing the Processing 1-9
Designing the Input 1-9
Structured Programming in RPG/400 1-9
Sequential Operation 1-9
Conditional Branching 1-10
Repeating an Operation 1-11
Do While 1-12
Do Until 1-14
Do 1-16
Summary of Structured Programming Operation Codes 1-18
Designing Applications 1-18
Single Program Design 1-18
Modular Program Design 1-19
Examples of Application Design 1-20

Chapter 2. Entering RPG/400 Specifications 2-1
The RPG/400 Specifications 2-1
The Control Specification 2-1
File Description Specifications 2-1
Extension Specifications 2-2
Line Counter Specifications 2-2
Input Specifications 2-2
Calculation Specifications 2-2
Output Specifications 2-2
Entering Your Program 2-3
Using SEU to Enter or Change an RPG/400 Source Program 2-3

Chapter 3. Compiling an RPG/400 Program 3-1
Create RPG/400 Program (CRTRPGPGM) Command 3-2
Using the CRTRPGPGM Command 3-3
Elements of the CRTRPGPGM Command Lines 3-4
Entering Elements from the CRTRPGPGM Command Display 3-4

Entering Only Certain Parameters 3-4
Entering Only the Parameter Values 3-5
CRTRPGPGM Command 3-5
Compiling under the System/38 Environment 3-14

Chapter 4. Error Messages, Testing and Debugging 4-1
Using, Displaying, and Printing Messages 4-1
Using Messages 4-1
SAA Flagging Messages 4-2
Displaying and Printing Messages 4-3
How to Run an RPG/400 Program 4-3
Using a Test Library 4-5
Using Breakpoints 4-7
Example of Using Breakpoints 4-7
Considerations for Using Breakpoints 4-11
Using a Trace 4-11
Example of Using a Trace 4-12
Considerations for Using a Trace 4-13
Using the DEBUG Operation Codes 4-13
Using the RPG/400 Formatted Dump 4-14
File Information Data Structure 4-24
File-Dependent Feedback Information. 4-31
Contents of File Information Data Structure After POST 4-31
Program Status Data Structure 4-34
Exception/Error Handling 4-36
File Exception/Errors 4-39
Exception/Error Indicators 4-40
File Information Data Structure (INFDS) 4-40
Status Codes 4-43
File Exception/Error Subroutine (INFSR) 4-46
Program Exception/Errors 4-51
Program Status Data Structure 4-51
Status Code 4-52
Program Exception/Error Subroutine 4-53

Chapter 5. The Detailed RPG Program Cycle 5-1
Detailed RPG/400 Program Cycle 5-3
RPG Exception/Error Handling Routine 5-8
Match Fields Routine 5-10
Overflow Routine 5-11
Look-Ahead Routine 5-11
Ending a Program without a Primary File 5-11
Program Control of File Processing 5-12

Chapter 6. General File Considerations 6-1
Device Independence/Device Dependence 6-1
Spooling 6-3
Output Spool 6-3
Externally Described/Program Described Files 6-4
Level Checking 6-7
File Locking by RPG/400 6-7

Record Locking by RPG/400 6-8
Unblocking Input Records and Blocking Output Records 6-8
Sharing an Open Data Path 6-9
Using the CL Command RCLRSC 6-9
RPG Specifications for Externally Described Files 6-10
File Description Specifications 6-10
Renaming Record Format Names 6-10
Ignoring Record Formats 6-12
Floating-Point Fields 6-13
Overriding or Adding RPG/400 Functions to the External Description . . 6-13
Output Specifications 6-15
Program Described Files 6-17
Printer Files 6-17
Page Overflow 6-18
Overflow Indicators 6-19
Fetch Overflow Logic 6-21
PRTCTL (Printer Control) Option 6-24
SEQ (Sequential) File 6-27
Special File 6-29
Primary/Secondary Multifile Processing 6-33
Multifile Processing with No Match Fields 6-34
Multifile Processing with Match Fields 6-34
Assigning Match Field Values (M1-M9) 6-35
Processing Matching Records 6-39

Chapter 7. Commitment Control 7-1
Using Commitment Control 7-1
Starting and Ending Commitment Control 7-1
Specifying Files for Commitment Control 7-2
Commitment Control Operations 7-2
Commitment Control Locks 7-3
Commitment Control in the Program Cycle 7-3
Example of Using Commitment Control 7-4

Chapter 8. Using DISK Files 8-1
Externally Described Disk Files 8-1
Record Format Specifications 8-1
Access Path 8-2
Valid Keys for a Record or File 8-6
Valid Search Arguments 8-6
Referring to a Partial Key 8-7
Processing Methods for Externally Described DISK Files 8-8
Program Described Disk Files 8-8
Indexed Files 8-8
Valid Search Arguments 8-8
Sequential Files 8-9
Record Address File 8-12
Limits Records 8-13
Relative Record Numbers 8-14
Externally Described File as Program Described 8-14
Methods for Processing Disk Files 8-14

Relative Record Number Processing 8-14
Consecutive Processing 8-15
Sequential by Key Processing 8-15
Random by Key Processing 8-21
Sequential within Limits Processing 8-22
Keyed Processing Examples 8-22
Valid File Operations 8-29

Chapter 9. Using WORKSTN files 9-1
Intersystem Communications Function (ICF) 9-1
Externally Described WORKSTN Files 9-1
Processing an Externally Described WORKSTN File 9-2
Function Key Indicators 9-4
Command Keys 9-5
Processing WORKSTN Files 9-5
Subfiles 9-7
Use of Subfiles 9-10
Program Described WORKSTN File 9-12
Program Described WORKSTN File with a Format Name 9-12
Output Specifications 9-13
Input Specifications 9-13
Calculation Specifications 9-14
Additional Considerations 9-14
Program Described WORKSTN File without a Format Name 9-14
Input File 9-14
Output File 9-15
Combined File 9-15
Multiple Device Files 9-15
WORKSTN File Examples 9-16
Sample Program 1—Inquiry 9-17
Sample Program 2—Data Entry with Master Update 9-23
Sample Program 3—Maintenance 9-30
Sample Program 4—WORKSTN Subfile Processing 9-40
Sample Program 5—Inquiry by Zip Code and Search on Name 9-48
Sample Program 6—Program Described WORKSTN File within FORMAT Name on Output Specifications 9-57
Sample Program 7—Variable Start Line 9-59
Sample Program 8—Read Operation with Time-Out 9-61

Chapter 10. Data Field Formats and Data Structures 10-1
Format of Fields in Files 10-1
Packed Decimal Format 10-1
Zoned Decimal Format 10-3
Binary Format 10-3
Signs 10-5
External Formats 10-5
Internal Format 10-6
Data Structures 10-6
Format of Data Structure Subfields in Storage 10-7
Data Structure Statement Specifications 10-8
Rules for Specifying Data Structure Statements 10-8

Multiple Occurrence Data Structure 10-9
Special Data Structures 10-9
Data Area Data Structure 10-9
File Information Data Structure (INFDS) 10-10
Program Status Data Structure 10-11
Data Structure Subfield Specifications 10-11
Rules for Subfield Specifications 10-12
Data Structure Examples 10-13

Chapter 11. Communicating with Objects in the System 11-1
Calling Other Programs 11-1
CALL (Call a Program) 11-3
PLIST (Identify a Parameter List) and PARM (Identify Parameters) . . . 11-4
Rules for Specifying PLIST 11-4
Rules for Specifying PARM 11-5
OS/400 Graphics Support 11-5
FREE (Deactivate a Program) 11-6
Calling Special Subroutines 11-6
Message-Retrieving Subroutine (SUBR23R3) 11-6
Moving Double-byte Data and Deleting Control Characters (SUBR40R3) 11-8
Moving Double-byte Data and Adding Control Characters (SUBR41R3) 11-10
Returning from a Called Program 11-11
A Normal End 11-11
An Abnormal End 11-12
Return without an End 11-12
Data Areas 11-13

Chapter 12. Using Indicators 12-1
Indicators Defined on RPG/400 Specifications 12-1
Overflow Indicators 12-2
Record Identifying Indicators 12-2
Rules for Assigning Record Identifying Indicators 12-3
Control Level Indicators (L1-L9) 12-4
Rules for Control Level Indicators 12-5
Split Control Field 12-10
Field Indicators 12-13
Rules for Assigning Field Indicators 12-13
Resulting Indicators 12-14
Rules for Assigning Resulting Indicators 12-14
Indicators not Defined on the RPG/400 Specifications 12-15
External Indicators 12-15
Internal Indicators 12-16
First Page Indicator (1P) 12-16
Last Record Indicator (LR) 12-16
Matching Record Indicator (MR) 12-17
Using Indicators 12-17
File Conditioning 12-17
Rules for File Conditioning 12-17
Field Record Relation Indicators 12-18
Assigning Field Record Relation Indicators 12-18
Function Key Indicators 12-21

Return Indicator (RT) 12-21
Halt Indicators (H1-H9) 12-22
Indicators Conditioning Calculations 12-22
Positions 7 and 8 12-22
Positions 9-17 12-23
Indicators Conditioning Output 12-27
Indicators Referred to as Data 12-32
*IN 12-32
*INxx 12-32
Additional Rules 12-32
Summary of Indicators 12-33

Chapter 13. Using Arrays and Tables 13-1
ARRAYS 13-1
Array Name and Index 13-1
The Essential Array Specifications 13-2
Coding a Run-Time Array 13-2
Loading a Run-Time Array 13-3
Array Information in One Record 13-3
Array Information in More Than One Record 13-3
Sequencing Run-Time Arrays 13-4
Coding a Compile-Time Array 13-4
Loading a Compile-Time Array 13-4
Rules for Array Input Records 13-5
Coding a Prerun-Time Array 13-7
Loading a Prerun-Time Array 13-8
Defining More than One Array 13-9
Two Run-Time Arrays 13-9
Mixing Compile-Time and Prerun-Time Arrays 13-9
Arrays in Alternating Format 13-9
Searching Arrays 13-10
Searching an Array without an Index 13-10
Searching an Array with an Index 13-13
Specifying Arrays 13-15
Specifying an Array in Calculations 13-15
Modifying Contents of Arrays 13-16
Adding Entries to Arrays 13-16
Array Output 13-17
Editing Entire Arrays 13-18
Examples of Using Arrays 13-18
TABLES 13-26
LOKUP with One Table 13-27
LOKUP with Two Tables 13-27
Specifying the Table Element Found in a LOKUP Operation 13-30

Chapter 14. Editing Numeric Fields 14-1
Edit Codes 14-1
Simple Edit Codes 14-2
Combination Edit Codes 14-2
User-Defined Edit Codes 14-4
Editing Considerations 14-4

Summary of Edit Codes 14-5
Edit Words 14-8
How to Code an Edit Word 14-9
Parts of an Edit Word 14-9
Forming the Body of an Edit Word 14-10
Forming the Status of an Edit Word 14-13
Formatting the Expansion of an Edit Word 14-14
Summary of Coding Rules for Edit Words 14-14
Formatting Edit Words 14-15
Editing Externally Described Files 14-15

Chapter 15. Alternate Collating Sequence and File Translation 15-1
Changing the Collating Sequence 15-1
Specifying an Alternate Collating Sequence 15-1
Filling Out the Alternate Collating Sequence Coding Sheet 15-1
Formatting the Alternate Collating Sequence Records 15-4
File Translation 15-4
Specifying a File Translation 15-5
Translating One File or All Files 15-5
Translating More than One File 15-6
Specifying the Files 15-6
Specifying the Table 15-6

Chapter 16. Auto Report Feature 16-1
Group Printing 16-1
Specifications 16-1
Examples 16-1
/COPY Statement Specifications 16-4
Modifying Copied Specifications 16-6
Modifying File Description Specifications 16-6
Modifying Input Field Specifications 16-7
Report Format 16-11
Spacing and Skipping 16-11
Placement of Headings and Fields 16-11
Page Headings 16-12
Reformatting *AUTO Page Headings 16-12
Body of the Report 16-13
Overflow of the D/T-*AUTO Print Lines 16-14
Generated Specifications 16-16
Generated Calculations 16-16
Generated Output Specifications 16-17
Programming Aids 16-22
Using CRTRPTPGM to Compile an Auto Report Source Program 16-26
Using the CRTRPTPGM Command 16-27
CRTRPTPGM Command 16-27
Examples of Using Auto Report 16-29
EXAMPLE 1-Sales Report 16-29
EXAMPLE 2-Sales Report with Three Levels of Totals 16-33
EXAMPLE 3-Sales Report with Group Indication 16-35
EXAMPLE 4-Sales Report with Cross-Column Totals 16-39
EXAMPLE 5-Sales Report using copied Specifications 16-42

EXAMPLE 6-Override Copied Input Specifications 16-46

Appendix A. RPG Compiler and Auto Report Program Service Information . A-1
Compiler Overview A-1
Compiler Phases A-2
Major Compiler Data Areas A-4
Compiler Error Message Organization A-4
Run-Time Subroutines A-4
Compiler Debugging Options A-5
*SOURCE Value for the OPTION Parameter A-6
*XREF Value for the OPTION Parameter A-6
*DUMP Value for the OPTION Parameter A-6
*LIST Value for the GENOPT Parameter A-6
*ATR Value for the GENOPT Parameter A-6
*XREF Value for the GENOPT Parameter A-6
*DUMP Value for the GENOPT Parameter A-6
*PATCH Value for the GENOPT Parameter A-6
*OPTIMIZE Value for the GENOPT Parameter A-6
ITDUMP Parameter A-7
SNPDUMP Parameter A-7
CODELIST Parameter A-7
PHSTRC Parameter A-7
Examples of Using Compiler Debugging Options A-7
IRP Layout A-25
Auto Report Program A-27

Appendix B. Sample Program B-1
Sample RPG Program (ORD400) B-1

Appendix C. Differences Between AS/400 RPG/400 and AS/400 RPG II System/36-Compatible C-1
Language Enhancements C-1

Appendix D. Data Communication D-1
Exception/Error Handling with ICF Files D-1
Communications Error Recovery D-2

Appendix E. RPG Restrictions E-1

Appendix F. Distributed Data Management (DDM) Files F-1

Appendix G. System/38 Environment Option of the RPG Compiler G-1
Differences Between System/38 RPG III and the System/38 Environment Option of the RPG Compiler G-1
Differences Between the System/38 Environment Option of the RPG Compiler and RPG/400 . . . . . G-1
File Types Supported By Each Compiler G-3

Appendix H. Glossary of Abbreviations H-1

Appendix I. Using Double Byte Character Set (DBCS) Data in RPG/400 Programs. I-1
Where You Can Use DBCS Data in RPG/400 Programs I-1
How to Work with DBCS Data in RPG/400 Programs I-1
Transparent Literals and Constants I-1
Additional Considerations for Using DBCS Data in RPG/400 Programs . . I-2
Example of Coding DBCS Data in an RPG/400 Program I-2

Glossary of Terms GLOSS-1

Index X-1

About This Manual

Who Should Use this Manual

This manual is a guide for the RPG/400 programming language on the AS/400[1] System using OS/400™. RPG/400 is a Systems Application Architecture™ (SAA)™ compiler, that adheres to SAA conventions. It is intended for people who have a basic understanding of data processing concepts and of the RPG/400 programming language. For a detailed description of RPG/400 specifications and operation codes, see the *Languages: RPG/400 Reference*

The topics this manual covers include:

- Design RPG/400 programs
- Code RPG/400 programs
- Enter and compile RPG/400 programs
- Test and debug RPG/400 programs
- Study coded RPG/400 examples and sample programs.

How This Manual is Organized

This manual is a user's guide to RPG/400. Chapter 1 introduces RPG/400. Chapters 2 to 5 describe coding, entering, compiling, testing, debugging, and putting a program into production. Chapters 6 to 15 describe how to use various elements such as files, indicators, arrays, and tables. The appendixes provide additional information on the System/38 Environment, compiler debugging options, and communications.

What You Should Know

Before you use this manual, you should be familiar with certain information:

- You should know how to describe files to the system and how to create jobs and output queues. This information is contained in *Programming: Data Management Guide*, SC21-9658.
- You should be familiar with your display station (also known as a work station), and its controls. There are also some elements of its display and certain keys on the keyboard that are standard regardless of which software system is currently running at the display station, or which hardware system the display station is hooked up to. Some of these keys are:
  - Cursor movement keys
  - Command keys
  - Field exit keys

[1] AS/400, OS/400, Systems Application Architecture, and SAA are trademarks of the International Business Machines Corporation.

- Insert and delete keys
- The Error Reset key.

This information is contained in *System Operations: Display Station User's Guide* SC21-9744.

- You should know how to operate your display station when it is hooked up to the IBM AS/400 System and running AS/400 software. This means knowing about OS/400 and the Control Language (CL) to do such things as:
- Sign on and sign off the display station
- Interact with displays
- Use Help
- Enter control commands
- Call utilities
- Respond to messages.

To find out more about this operating system and its control language, refer to these IBM publications:

- *Programming: Control Language Reference*, SBOF-0481
- *Programming: Control Language Programmer's Guide*, SC21-8077
- *Programming: Command Reference Summary*, SC21-8076
- *Programming: System Reference Summary*, SC21-8104

- You should know how to call and use certain utilities available on the AS/400 System:
  - The Screen Design Aid (SDA) utility used to design and code displays. This information is contained in *Application Development Tools: Screen Design Aid User's Guide and Reference*, SC09-1171.
  - The Source Entry Utility (SEU), which is a full-screen editor you can use to enter and update your source and procedure members. This information is contained in *Application Development Tools: Source Entry Utility User's Guide and Reference*, SC09-1172.
- You should know how to interpret displayed and printed messages. This information is contained in "Using, Displaying, and Printing Messages" on page 4-1 in Chapter 4, "Error Messages, Testing and Debugging."
- You should be familiar with the RPG/400 program cycle, how indicators affect the program cycle, and how to code entries on the RPG/400 specification sheets.

  These general items about RPG/400 are taught in an RPG/400 coding class. Detailed information on RPG/400 can be found in *Languages: RPG/400 Reference*, SC09-1089.

If You Need More Information

You might need to refer to other AS/400 manuals for specific information about a particular topic. They are listed below:

- *Information Directory*, GC21-9678, which contains a brief description of each manual in the AS/400 library and information on how to order additional publications.

- *Licensed Programs Installation Guide*, SC21-9765, which describes how to install RPG/400 on your system.
- *System Operations: Operator's Guide*, SC21-8082, which describes how to operate the AS/400 System.
- *Programming: Data Description Specifications Reference*, SC21-9620, which describes data description specifications that are used for describing files.
- *Communications: Distributed Data Management User's Guide*, SC21-9600, which contains information about remote communication for the RPG/400 programmer.
- *Programming: Data Base Guide*, SC21-9659, which contains a detailed discussion of the AS/400 data base structure. This manual also describes how to use data description specifications (DDS) keywords.
- *Communications: Programmer's Guide*, SC21-9590, which provides information an application programmer needs to write applications that use AS/400 communications and the Intersystem Communications Function file.
- *Programming: Graphical Data Display Manager Programming Reference*, SC33-0537, and *Programming: Graphical Data Display Manager Programming Guide*, SC33-0536, which provide guidance on the Graphical Data Display Manager (GDDM)[2] for programmers who need to write graphics applications.
- *Programming: Structured Query Language/400 Programmer's Guide*, SC21-9609 and *Programming: Structured Query Language/400 Reference*, SC21-9608 for information on SQL.
- *System/38 Environment Programmer's Guide and Reference*, SC21-9755, which describes migrating from System/38 and converting to AS/400 System.
- *Programming: Data Management Guide*, SC21-9658, which contains information about managing key aspects of the system.

---

[2] GDDM is a trademark of International Business Machines Corporation.

Entering Only the Parameter Values

You have the choice of entering only the parameter values without specifying the parameter keywords. Since, in this case, there is no keyword to tell the system which value belongs to which parameter, you must enter all the values in the sequence shown below. You need not enter the entire set of options, but you must enter the options for all the parameters up to the one you are interested in. The system will use the default values for the remaining parameters.

For example, to compile a source program in file QRPGSRC in member ABC in library SRCLIB enter:

```
CRTRPGPGM QTEMP/ABC SRCLIB/QRPGSRC *PGM
```

Notice that you also had to enter names for the program and library for the compiled program. The system recognizes which option belongs to which parameter by the position of the value on the compiler command line. You can enter a maximum of three parameter values positionally.

For more information see *Programming: Control Language Reference*. This information applies only to native commands. For System/38 Environment commands, look in the appendixes of this book and System/38 manuals.

CRTRPGPGM Command

Following are examples of the prompt screens for the CRTRPGPGM command. The example screens are provided in sets. The first screen in the set describes the values you can enter, the second screen presents the keywords and defaults. You can toggle between the values and keywords screens by pressing F11. The text which follows the screens describes those keywords and defaults.

In the description of the parameters all defaults are explained first and highlighted. The parameters are presented in sequence. Follow this sequence if you are entering only the parameter values without the corresponding parameter acronym.

Note: For a description of the differences between compiling RPG/400 and System/38 Environment RPG III see Appendix G, "System/38 Environment Option of the RPG Compiler."

```
CRTRPGPGM                Create RPG/400 Program

Type choices, press Enter.

Program . . . . . .  *CTLSPEC__   Name, *CTLSPEC
  Library . . . . . .   *CURLIB___  Name, *CURLIB
Source file . . . . . QRPGSRC___   Name, QRPGSRC
  Library . . . . . .   *LIBL_____ Name, *LIBL, *CURLIB
Source member . . . . *PGM_______  Name, *PGM
Generation severity level . . . 9___       0-99
Text 'description' . . . . . *SRCMBRTXT_____________________________
_____

                      Additional Parameters

Source listing options . . . . .  ________    *SOURCE, *NOSOURCE, *SRC...
            + for more values     ________
Generation options . . . . . .    __________  *LIST, *NOLIST, *XREF...
            + for more values     __________
SAA flagging . . . . . .  *NOFLAG       *NOFLAG, *FLAG
                                                              More...

F3=Exit   F4=List   F5=Refresh   F11=Keywords   F12=Previous
F13=How to use this display
```

```
CRTRPGPGM                Create RPG/400 Program

Type choices, press Enter.

Program . . . . . . PGM          *CTLSPEC__
  Library . . . . . .              *CURLIB___
Source file . . . . . SRCFILE      QRPGSRC___
  Library . . . . . .              *LIBL_____
Source member . . . . SRCMBR       *PGM_______
Generation severity level . . . GENLVL     9___
Text 'description' . . . . . TEXT          *SRCMBRTXT_____________________
_____________

                      Additional Parameters

Source listing options . . . . . OPTION       ________
                    + for more values         ________
Generation options . . . . . GENOPT           __________
                    + for more values         __________
SAA flagging . . . . . SAAFLAG       *NOFLAG
                                                              More...

F3=Exit   F4=List   F5=Refresh   F11=Choices   F12=Previous
F13=How to use this display
```

Figure 3-2. First Set of CRTRPGPGM Prompt Screens

PGM

Specifies the library and program name by which the compiled RPG/400 program is to be known. If no library is specified, the created program is stored in the current library.

***CTLSPEC**
The program name specified in positions 75 through 80 of the control specification is used.

If the program name is not specified on the control specification, but the source program is from a data base file, the member name, specified by the SRCMBR parameter, is used as the program name. If the source is not from a data base file, the program name defaults to RPGOBJ.

program-name
Enter the name by which the program will be known.

***CURLIB**
The compiled program is stored in the current library. If you have not specified a current library, QGPL will be used.

library-name
Enter the name of the library where the compiled program is to be stored.

SRCFILE
Specifies the name of the source file that contains the RPG/400 source program to be compiled and the library where the source file is located.

QRPGSRC
The default source file, QRPGSRC, contains the RPG/400 source program to be compiled.

source-file-name
Enter the name of the source file that contains the RPG/400 source program to be compiled.

***LIBL**
The system searches the library list to find the library where the source file is located.

***CURLIB**
The current library will be used to find the source file. If you have not specified a current library, QGPL will be used.

library-name
Enter the name of the library where the source file is stored.

SRCMBR
Specifies the name of the member of the source file that contains the RPG/400 source program to be compiled. This parameter can be specified only if the source file named in the SRCFILE parameter is a data base file.

*PGM
Use the name specified by the *PGM parameter as the source file member name. The compiled program will have the same name as the source file member. If no program name is specified by the *PGM parameter, the command uses the first member created in or added to the source file as the source member name.

source-file-member-name
Enter the name of the member that contains the RPG/400 source program.

GENLVL:
Specifies whether or not a program object is generated depending on the severity of the errors encountered. A severity level value corresponding to the severity level of the messages produced during compilation can be specified with this parameter. If errors occur in a program with a severity level equal to or greater than the value specified in this parameter, the compilation is ended. For example, if you do not want a program generated if you have messages with a severity level of 30 or greater, specify 30 in this parameter.

9
A program object will not be generated if you have messages with a severity level greater than or equal to 9. If a severity level greater than 9 is specified, the program may contain errors that can cause unpredictable results when the program runs.

severity-level-value:
Enter a number, 0 through 99.

Note: The severity level value of RPG/400 messages does not exceed 50.

TEXT
Lets the user enter text that briefly describes the program and its function. The text appears whenever program information is displayed.

*SRCMBRTXT
The text of the source member is used.

*BLANK
No text appears.

'*description*'
Enter the text that briefly describes the program and its function. The text can be a maximum of 50 characters and must be enclosed in apostrophes. The apostrophes are not part of the 50-character string. Apostrophes are not required if you are entering the text on the prompt screen.

OPTION
Specifies the options to use when the source program is compiled. You can specify any or all of the options in any order. Separate the options with a blank space.

***SOURCE**
Produces a source listing, consisting of the RPG/400 program input and all compile-time errors.

***NOSOURCE**
A source listing is not produced. If *NOSOURCE is specified, the system defaults to *NOXREF.

The acceptable abbreviation for *SOURCE is *SRC and for *NOSOURCE is *NOSRC.

***XREF**
Produce a cross-reference listing and key field information table (when appropriate) for the source program.

***NOXREF**
A cross-reference listing is not produced.

*NOXREF is the default when either *NOSOURCE or *NOSRC is specified.

***GEN**
Create a program object that can be run after the program is compiled.

***NOGEN**
Do not create a program object.

***NODUMP**
Do not dump major data areas when an error occurs during compilation.

***DUMP**
Dump major data areas when an error occurs during compilation.

***NOSECLVL**
Do not print second-level message text on the line following the first-level message text.

***SECLVL**
Print second-level message text.

GENOPT

Specifies the options to use to create the program object: the printing of the IRP (intermediate representation of a program), a cross-reference listing of objects defined in the IRP, an attribute listing from the IRP, and the program template. GENOPT reserves a program patch area, and specifies optimization of a program for more efficient running. These results may be useful if a problem occurs when you are trying to run the compiled program. You can specify any or all of the options in any order. Separate the values with a blank. For a description of the GENOPT parameter and the information it provides, see "Compiler Debug-

ging Options" on page A-5 in Appendix A, "RPG Compiler and Auto Report Program Service Information."

*NOOPTIMIZE
Do not process program optimization.

*OPTIMIZE
Process program optimization. With *OPTIMIZE the compiler generates a program for more efficient processing and one that will possibly require less storage. Specifying *OPTIMIZE can substantially increase the time required to create a program. Existing program objects may be optimized with the CL command CHGPGM.

SAAFLAG

Specifies if there will be flagging of specifications not supported by SAA RPG For more information on SAA flagging, how and why to use it, see "Using, Displaying, and Printing Messages" on page 4-1.

*NOFLAG
No flagging will be performed.

*FLAG
Flagging will be performed. Messages will be listed under the heading of SAA Message Summary. No SAA message will be issued for a specification if a message of severity 30 or above is issued for that specification.

```
CRTRPGPGM                Create RPG/400 Program

Type choices, press Enter.

Print file . . . . . .  QSYSPRT___    Name
  Library . . . . . .     *LIBL_____  Name, *LIBL, *CURLIB
Replace program . . . .  *YES         *YES, *NO
User profile . . . . . .  *USER_      *USER, *OWNER
Authority . . . . . .    *CHANGE___   Name, *CHANGE, *ALL, *USE...
Phase trace . . . . . .  *NO_         *NO, *YES
Intermediate text dump . . . . .  *NONE_____________________________
_____
Snap dump . . . . . .    *NONE_____________________________
_____
Codelist . . . . . .     *NONE_____________________________
_____
Ignore decimal data error . . .  *NO_        *NO, *YES

                                                                Bottom
F3=Exit   F4=List   F5=Refresh   F11=Keywords   F12=Previous
F13=How to use this display
```

```
CRTRPGPGM                Create RPG/400 Program

Type choices, press Enter.

Print file . . . . . . PRTFILE       QSYSPRT___
  Library . . . . . .                  *LIBL_____
Replace program . . . . REPLACE      *YES
User profile . . . . . . USRPRF      *USER_
Authority . . . . . . AUT            *CHANGE___
Phase trace . . . . . . PHSTRC       *NO_
Intermediate text dump . . . . . ITDUMP      *NONE_____________________
_____
Snap dump . . . . . . SNPDUMP        *NONE_____________________
_____
Codelist . . . . . . CODELIST        *NONE_____________________
_____
Ignore decimal data error . . . IGNDECERR    *NO_

                                                                Bottom
F3=Exit   F4=List   F5=Refresh   F11=Choices   F12=Previous
F13=How to use this display
```

Figure 3-3. Second Set of CRTRPGPGM Prompt Screens

PRTFILE

Specifies the name of the file where the compiler listing is to be placed and the library where the file is located. If you specify a file whose record length is less than 132, information will be lost.

QSYSPRT

If a file name is not specified, the compiler listing is placed in the IBM-supplied file, QSYSPRT. The file QSYSPRT has a record length of 132.

file-name
Enter the name of the file where the compiler listing is to be placed.

***LIBL**
The system searches the library list to find the library where the file is located.

***CURLIB**
The current library will be used to find the file. If you have not specified a current library, QGPL will be used.

library-name
Enter the name of the library where the file is located.

REPLACE
Specifies if a new program object will be created without first deleting any existing program object.

***YES**
A new program object will be created and any existing program object of the same name in the specified library will be moved to library QRPLOBJ.

***NO**
A new program object will not be created if a program object of the same name already exists in the specified library.

USRPRF
Specifies the user profile the compiled RPG/400 program runs under. This profile controls which objects can be used by the program (including what authority the program has for each object).

***USER**
The program runs under the user profile of the program's user.

***OWNER**
The program runs under the user profiles of both the program's owner and user. The collective sets of object authority in both user profiles are used to find and access objects while the program is running. Any objects that are created during the program are owned by the program's user.

Note: The USRPRF parameter reflects the security requirements of your installation. The security facilities available on the AS/400 System are described in detail in the *Security Concepts and Planning* and the *Programming: Control Language Reference.*

AUT
Specifies what authority for the program and its description is being granted to the public. The authority can be altered for all or for specified users after

program creation with the CL commands GRTOBJAUT (Grant Object Authority) or RVKOBJAUT (Revoke Object Authority). (For further information on these commands, see the *Programming: Control Language Reference.*)

***CHANGE**
The public has only operational rights for the compiled program. Any user can run the program, but cannot change it.

***USE**
The public can run the compiled program, but cannot debug or change it.

***ALL**
The public has complete authority for the program.

***EXCLUDE**
The public cannot use the program.

authorization-list name
Name of an authorization list to which the program is added. For a description of the authorization list and how to create it see the *Programming: Control Language Reference.*

Note: Use the AUT parameter to reflect the security requirements of your installation. The security facilities available are described in detail in the *Security Concepts and Planning.*

PHSTRC
Specifies whether information about compiler phases is provided on the listing. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

***NO**
Do not provide information about compiler phases.

***YES**
Provide information about compiler phases.

ITDUMP
This parameter specifies whether a dynamic listing of intermediate text for one or more specified phases is to be printed at compile time as each IT record is being built. This parameter also specifies whether a flow of the major routines run in one or more specified phases is to be printed. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

***NONE**
No intermediate dump is produced.

phase-name
Enter the last two characters of phase name. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

SNPDUMP
Specifies whether the major data areas are to be printed after the running of one or more specified phases. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

***NONE**
No snap dump is produced.

phase-name
Enter the last two characters of phase name. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

CODELIST
Specifies whether a dynamic listing of the IRP is to be printed during compilation of one or more specified phases of the source program. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

***NONE**
Do not produce a code listing for each of the code generating phases run.

***ALL**
Produce a code listing for each of the code generating phases run.

phase-name
Enter the last 2 characters of phase name. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

IGNDECERR
Specifies whether decimal data errors detected by the system are ignored by the program.

***NO**
Do not ignore decimal data errors. The result of the operation being processed when the error occurs is unknown. The compiler generates an error message on the compiler listing to notify the user that this option was specified. Incorrect results that occur while a program is running when this option is specified are the user's responsibility.

***YES**
Ignore decimal data errors.

Compiling under the System/38 Environment

You can also compile an RPG/400 source program from the System/38 Environment. Here, you would call the compiler with the same commands as you would use in the AS/400 Native Environment (CRTRPGPGM to call up the RPG/400 compiler, and CRTRPTPGM to call up the auto report). To compile a program from the System/38

Environment, use the CL command `CALL QCL` to call up the System/38 Environment before entering the `CRTRPGPGM` command.

For more information on the differences between RPG/400 in AS/400 Native Environment and in the System/38 Environment see Appendix G, "System/38 Environment Option of the RPG Compiler."

For further information about programming in the System/38 Environment, refer to the *System/38 RPG III Reference Manual and Programmer's Guide.*

Chapter 4. Error Messages, Testing and Debugging

This chapter describes error messages you may receive from RPG, how to interpret them, and how they can be displayed and printed. It also describes testing and debugging an RPG/400 program using functions provided by RPG/400 and OS/400.

OS/400	RPG
• Test library • Breakpoints • Traces	• DEBUG operation code • DUMP operation code

The OS/400 functions allow you to use CL commands to test programs while protecting your production files, and let you observe and debug operations as a program runs. No special source code is required to use the OS/400 functions. The RPG/400 functions can be used independently of the OS/400 functions or in combination with them to:

- Debug a program, or
- Produce a formatted dump of indicator settings and the contents of fields, data structures, arrays, and tables.

Special source code is required to use the RPG/400 DEBUG and DUMP operation codes. A formatted dump can also be obtained by a user's response to a run-time message.

A file information data structure and a program status data structure can provide additional debugging information. These data structures are described later in this chapter. Following this is information on exception/error handling.

Using, Displaying, and Printing Messages

Using Messages

This manual refers to messages you receive during both compile and run-time. These messages are displayed on your screen or printed on your compiler listing. There are no message manuals for this product.

Each message contains a minimum of three parts as illustrated below:

```
A 10
B Syntax of Program-Identification entry is invalid.  Defaults to RPGOBJ.
C Cause:The Program-Identification entry (positions 75-80) of a control
  specification has an invalid syntax: the first character is not
  alphabetic or it is not left-justified, or it contains embedded blanks
  or special characters.  Defaults to RPGOBJ.
  Recovery:Specify RPGOBJ or a valid entry (positions 75-80) for the
  Program-Identification option.  Recompile.
```

A A number indicating the severity of the error. Numbers 40 and above are reserved for run-time errors.

Severity	Meaning
00	An informational message displayed during entering, compiling, and running. This level is used to convey information to the user. No error has been detected and no corrective action is necessary.
10	A warning message displayed during entering, compiling and running. This level indicates that an error was detected but is not serious enough to interfere with the running of the program. The results of the operation are assumed to be successful.
20	An error message displayed during compiling. This level indicates that an error was made, but the compiler is taking a recovery that might yield the desired code. The program may not work as the author intended.
30	A severe error message displayed during compiling. This level indicates that an error too severe for automatic recovery was detected. Compilation is completed, but running the program cannot be attempted.
40	An abnormal end of program or function message displayed during running. This level indicates an error that forces cancellation of processing. The operation may have ended because it was unable to handle valid data, or possibly because the user cancelled it.
50	An abnormal end of job message displayed during running. This level indicates an error that forces cancellation of job. The job may have ended because a function failed to perform as required, or possibly because the user cancelled it.
99	A user action to be taken during running. This level indicates that some manual action is required of the operator, such as entering a reply, changing diskettes, or changing printer forms.

B The text you see online or on a listing. This text is a brief, generally one sentence, description of the problem.

C The text you see online when you press F4 from the screen with the first-level text. This text will be printed on your listing if you specify *SECLVL in your compile-time options. The IBM-supplied default for this option is *NOSECLVL. This text contains an expanded description of the message and a section detailing the correct user response.

You can enter D to obtain RPG formatted dump, S to obtain system dump, C to cancel, or G to continue processing at *GETIN.

SAA Flagging Messages

In addition to the messages described above, RPG/400 also has a set of messages which flag those RPG/400 features not supported by SAA RPG. These messages are requested with a compiler option, SAAFLAG, described in "CRTRPGPGM Command" on page 3-5. The default value for this option is *NOFLAG. If you select *FLAG you will find these messages printed separately under the heading SAA

Message Summary. The SAA flagging messages are to help the programmer when writing portable code. If you are seeking maximum portability, you should eliminate the flagged codes from your program. A program which has only SAA messages will compile and run correctly on the AS/400 system. SAA messages are informational messages only. Severe error messages may cause SAA messages to be suppressed.

SAA messages are divided in the same way as the other messages described here. A sample message is:

```
A 0
B SAA RPG does not support numeric fields with more than 15 digits.
C Cause:Systems Application Architecture Common Programming Interface RPG
  does not support numeric fields with more than 15 digits.
  Recovery:If SAA RPG adherence is required, change the program and
  recompile.
```

These messages flag only RPG/400 specific functions.

Displaying and Printing Messages

To display or print a particular message or messages, use the DSPMSGF or DSPMSGD commands. These commands are described in the *Programming: Control Language Reference*. The library name for your product is QRPG, and the file names are QRPGMSG and QRPGMSGE. The run-time messages are stored in a file called QRPGMSGE in library QSYS. The compile time messages are in file QRPGMSG in library QRPG.

Note: If you have any comments or suggestions concerning the messages, please use the Reader Comment Form included with this manual and send them to us.

How to Run an RPG/400 Program

There are many ways to run an RPG/400 program depending on how the program is written and who is using the program. See the *CL Programmer's Guide* for the various ways to run an RPG/400 program. The three most common ways of running an RPG/400 program are through:

- A high-level language CALL statement or operation
- An application-oriented menu
- A user-created command.

The CL statement CALL can be part of a batch job, it can be entered interactively by a work station user, or it can be included in a CL program. An example is CALL PAYROLL. PAYROLL is the name of either a CL program or an RPG/400 program that is called and then run. An RPG/400 program can call another program with the CALL operation code (see Chapter 11, "Communicating with Objects in the System").

Another way to run an RPG/400 program is through an application-oriented menu. The work station user can request an application-oriented menu and then select an

option that will call the appropriate program. Figure 4-1 is an example of an application-oriented menu:

```
                    PAYROLL DEPARTMENT MENU

1.   Inquire into employee master

2.   Change employee master

3.   Add new employee

4.   Return


Option:_
```

Figure 4-1. Example of an Application-Oriented Menu

This menu is normally written as a CL program in which each option calls a separate RPG/400 program. When an RPG/400 program is ended, the system returns control to the calling program or to the user. This could be a work station user, a CL program (such as the menu handling program), or another RPG/400 program.

You can also create a command yourself to run an RPG/400 program by using a command definition. (See the *Programming: Control Language Programmer's Guide* for a description of how to define a command.) For example, you can create a command, PAY, which calls a PAYROLL program. A user-created command can be entered into a batch job, or it can be entered interactively by a work station user.

The OS/400 functions allow you to use CL commands to test programs while protecting your production files, and let you observe and debug operations as a program runs. No special source code is required to use the OS/400 functions. The RPG/400 functions can be used independently of the OS/400 functions or in combination with them to:

- Debug a program, or
- Produce a formatted dump of indicator settings and the contents of fields, data structures, arrays, and tables.

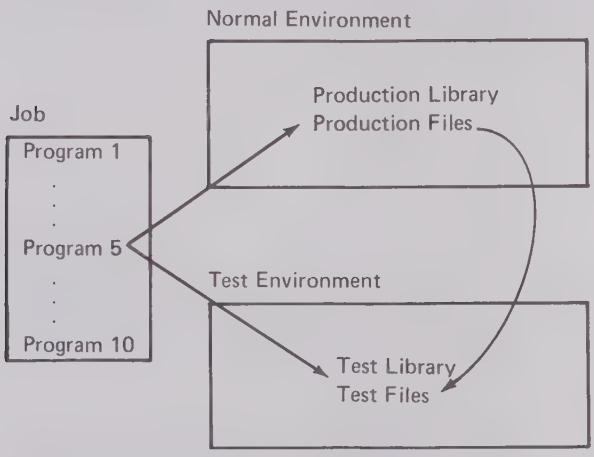
Special source code is required to use the RPG/400 DEBUG and DUMP operation codes. A formatted dump can also be obtained by a user's response to a run-time message.

A file information data structure and a program status data structure can provide additional debugging information. These data structures are described later in this chapter.

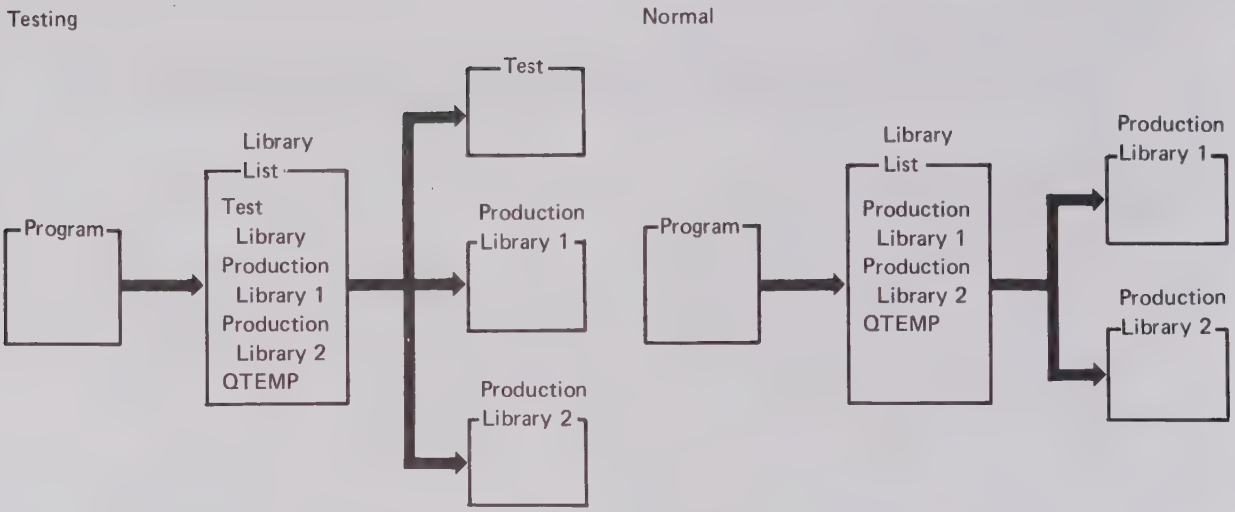
Using a Test Library

The basic concept of testing and debugging is that of a separate testing environment. Programs running in a normal operating environment can read, update, and write records that are in either test or production libraries. Programs running in a test environment can read, update, and write records in either test or production libraries. However, to prevent data base files in production libraries from being modified unintentionally, you can specify the UPDPROD(*NO) option on the CL command STRDBG (Start Debug) (see the *CL Reference*).

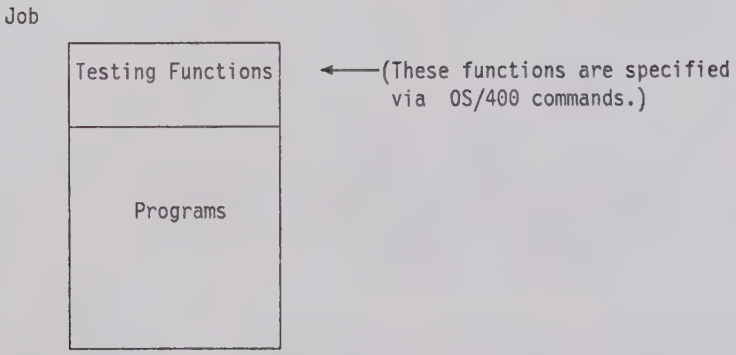
On the AS/400 system, you can copy production files into the test library or you can create special files for testing in this library. A test copy of a file and its production copy can have the same name if the files are in different libraries. You can use the same file name in the program for either testing or normal processing.



For testing, you must place the test library name ahead of the production library name in the library list for the job that contains the program to be tested. For normal running, the production library should be the only library named in the library list for that job (that is, the test library should not be named).



No special statements for testing are contained within the program being tested. The same program being tested can be run normally without modifications. All testing functions are specified within the job that contains the program and not within the program.



Testing functions apply only to the job in which they are specified. A program can be used concurrently in two jobs: one job that is in a test environment and another job that is in a normal processing environment.

OS/400 testing functions let you interact with a program as it runs to observe its processing. These functions include using breakpoints and traces.

Using Breakpoints

Breakpoints can be used to stop your program at a specified breakpoint. This breakpoint can be a statement number or a label in your program. If you use a label as a breakpoint rather than a statement number, the label can be:

- On a TAG statement in the program
- Associated with a step in the RPG/400 program cycle (for example, *TOTC indicates the beginning of total calculations and *TOTL indicates the beginning of total output)
- Associated with a function done by your RPG/400 program (for example, SQRT indicates the square root function).

When a breakpoint is encountered for an interactive job, the system displays the breakpoint at which the program has stopped and, if requested, the values of program variables. After you get this information (in a display) you can go to a Command Entry Screen and then enter CL commands to request other functions (such as displaying or changing a variable, adding a breakpoint, or adding a trace).

When a breakpoint is encountered for a batch job, a breakpoint program can be called. You must create this breakpoint program to handle the breakpoint information.

Example of Using Breakpoints

Figure 4-2 shows a source listing of a sample RPG/400 program, DBGPGM, and the CL commands that add breakpoints at statements 1200 and 1500. The value of variable *IN is displayed when the breakpoint at statement 1200 is reached, and the value of variables FLD1 and PART are displayed when the breakpoint at statement 1500 is reached.

CL Commands

```
STRDBG  PGM(EXAMPLES/DBGPGM)
ADDBKP  STMT(1200) PGMVAR((*IN))
ADDBKP  STMT(1500) PGMVAR((FLD1) (PART))
```

All CL commands are explained in the *Programming: Control Language Reference.*

```
5728RG1 R01 M00 880715          IBM AS/400 RPG/400          QGPL/DBGPGM      01/15/88  13:42:19     Page       2
SEQUENCE                                                                IND   DO   LAST       PAGE   PROGRAM
NUMBER   *...1....+....2....+....3....+....4....+....5....+....6....+....7...*  USE   NUM  UPDATE     LINE   ID
                        S o u r c e   L i s t i n g
         H                                                                                      *****
     100 FTESTX   IF  F       5           DISK                                                01/01/88
     200 FTESTA   UF  F      10           DISK                                                01/01/88
     300 ITESTX   NS  01                                                                      01/01/88
     400 I                                     1   5 PART                                     01/01/88
     500 ITESTA   NS  02                                                                      01/01/88
     600 I                                     1   5 FLD1                                     01/01/88
     700  ****************************************************                                01/01/88
     800  *       MAINLINE                                                                    01/01/88
     900  ****************************************************                                01/01/88
    1000 C           LOOP      TAG                                                          01/01/88
    1100 C                     READ TESTX                    66                   3         01/01/88
    1200 C   66                GOTO ENDPGM                                               01/01/88
    1300 C                     READ TESTA                    67                   3         01/01/88
    1400 C  N67                MOVE PART      FLD1                                       01/01/88
    1500 C  N67                EXCPTMAST                                                 01/01/88
    1600 C  N66                GOTO LOOP                                                 01/01/88
    1700 C           ENDPGM    TAG                                                       01/01/88
    1800 C                     SETON                     LR                  1           01/01/88
    1900 OTESTA  E                MAST                                                   01/01/88
    2000 O                        FLD1       5                                          01/01/88
           * * * * *  E N D   O F   S O U R C E  * * * * *
```

Figure 4-2. Sample RPG/400 Program DBGPGM

The first breakpoint is used to tell you where you are in the program. Figure 4-3 shows the display as a result of reaching the first breakpoint.

```
                        Display Breakpoint

 Statement/Instruction . . . . . :   1200 /004A
 Program . . . . . :   DBGPGM
 Recursion level . . . . . :   1
 Start position  . . . . . :   1
 Format  . . . . . :   *CHAR
 Length  . . . . . :   *DCL


 Variable  . . . . . :   *IN
   Lower/upper bounds  . . . . . :   (1:99)
   Type  . . . . . :   CHARACTER
   Length  . . . . . :   1
   Element     -------------------- Values ------------------- .
          1  '1'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         11  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         21  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         31  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'             +

Press Enter to continue.

F3=Exit program   F10=Command entry
```

```
                        Display Breakpoint

 Statement/Instruction . . . . . :   1200 /004A
 Program . . . . . :   DBGPGM
 Recursion level . . . . . :   1
 Start position  . . . . . :   1
 Format  . . . . . :   *CHAR
 Length  . . . . . :   *DCL


         41  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         51  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         61  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         71  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         81  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         91  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'

Press Enter to continue.

F3=Exit program   F10=Command entry
```

Figure 4-3. First Breakpoint Display for DBGPGM

Figure 4-4 shows the display as a result of reaching the second breakpoint.

Appendix G. System/38 Environment Option of the RPG Compiler

The System/38 Environment Option of the RPG Compiler compiler supports the same RPG syntax as the System/38 RPG III compiler, and the System/38 object naming conventions. The rest of this appendix discusses differences between System/38 RPG and System/38 Environment Option of the RPG Compiler, differences between System/38 Environment Option of the RPG Compiler and AS/400 RPG/400, and the file types supported by each compiler.

Differences Between System/38 RPG III and the System/38 Environment Option of the RPG Compiler

The System/38 Environment Option of the RPG Compiler support is different from System/38 RPG III in the following ways.

- The source member name on the create command is used for the name of the spooled output file that contains the compiler output.
- The format of the date used when the program is run is in the format described in the job value (set by the job description or by the CHGJOB command) rather than the system value.
- Card devices are not supported. If the RPG source specifies any of the card device syntax, an error of severity 30 will occur when you compile the program.
- Listing format differs.
- Message format is different for both compile and run time messages.

For those items that are the same as System/38 RPG III support, see the *System/38 RPG III Reference Manual and Programmer's Guide*, SC21-7725.

Differences Between the System/38 Environment Option of the RPG Compiler and RPG/400

Use RPG38 as the source type for the member containing the RPG source statements. The programs created will have the same values in the object attribute.

If you are using the CRTRPGPGM or CRTRPTPGM command directly be sure to use the command in the QSYS38 library.

Most of the information in this User's Guide is applicable to the System/38 Environment Option of the RPG Compiler support, with the exceptions in the following list.

- File and program names must follow the System/38 naming convention (object.library) on the /COPY statement and the FREE and CALL operation codes.

- The format of the information returned from a POST operation to a specific device is the same as the support on System/38.
- Numerics are not supported for the MOVEA operation code.
- The create commands are the same as the System/38.

This table shows the difference between the RPG/400 and the System/38 Environment Option of the RPG Compiler environments.

RPG/400 Parameter	System/38 Environment Option of the RPG Compiler Parameter	RPG/400 Options	System/38 Environment Option of the RPG Compiler Options	Comments
REPLACE	N/A			new parameter
		*YES	N/A	new option
		*NO	N/A	new option
AUT	PUBAUT			AUT replaces PUBAUT
		*CHANGE	*NORMAL	*CHANGE replaces *NORMAL
		*EXCLUDE	*NONE	*EXCLUDE replaces *NONE
		*USE	N/A	new option
		authori-zation- list-name	N/A	new option
PGM	PGM			Existing Parameter
		*CURLIB	N/A	new option/new default
SRCFILE	SRCFILE			Existing Parameter
		*CURLIB	N/A	new option
PRTFILE	PRTFILE			Existing Parameter
		*CURLIB	N/A	new option
OPTION	OPTION			Existing Parameter
		*SECLVL	N/A	new option
		*NOSECLVL	N/A	new option
SAAFLAG	N/A			new Parameter
		*NOFLAG	N/A	new option
		*FLAG	N/A	new option

When you convert a S/38-compatible program to an AS/400 program, you can, and in some cases, must, use the support as documented in this manual for the above list of items.

File Types Supported By Each Compiler

Similar to programs, files (object type *FILE) can be created with a System/38 or an OS/400 attribute. System/38 files have a 38 appended to their object attribute. OS/400 files do not. See the *S/38 Environment Programmer's Guide/Reference* for information on object attributes for the various types of files.

Any type of RPG/400 program can use files created with either attribute. System/38 Environment Option of the RPG Compiler programs are not restricted to using System/38 files. AS/400 RPG/400 programs are not restricted to using OS/400 files. For example, a System/38 Environment Option of the RPG Compiler program can use an OS/400 display file or data base file. An AS/400 RPG/400 program can use a System/38 display file or data base file.

The same ability to mix file types and program types applies to the communications file types, even though the System/38 file types are different from the OS/400 file types. An System/38 Environment Option of the RPG Compiler program can use an ICF file. An AS/400 program can use a communications, BSC or mixed device file. There are some items worth noting about such combinations. A System/38 Environment Option of the RPG Compiler program that uses an ICF file uses the System/38 Environment Option of the RPG Compiler *STATUS values. Also, the System/38 format and content of the information returned from a POST operation to a specific device is used, which will not return all the information available for an ICF file. An AS/400 program that uses a BSC, communications or mixed device file uses the AS/400 RPG *STATUS values, some of which are set based on major/minor return codes. Also, the format of the information returned from a POST operation to a specific device is the AS/400 RPG/400 version. Some of the items, such as remote location name, will not be returned, since those file types do not support remote location name.

QRGIC compiler phase A-3
QRGMC compiler phase A-3
QRGRT compiler phase A-3
QRGSC compiler phase A-3
QRGSE compiler phase A-3
QRGSF compiler phase A-3
QRGSI compiler phase A-3
QRGSO compiler phase A-3
QRGYC compiler phase A-3
QRG1 compiler phase A-3
QRPT0000 A-27
QRPT0001 A-27
QRPT0002 A-27
QRPT0003 A-27
QRPT0004 A-27
QRPT0005 A-27
QRPT0008 A-27

R

random by key processing
  description 8-21
  example 8-23
random by relative record number processing 8-14
RCLRSC 6-9
READ (read a record) operation code
  with data communication D-1
READC (read next modified record) operation code
  with WORKSTN subfile 9-9
RECNO
  with relative record number processing 8-14
record
  limits 8-13
  releasing 6-8
  renaming 6-10
record address file
  relative record number 8-12
  sequential-within-limits 8-12
  with limits records 8-13
  with relative record numbers 8-12
record address limits file
  See record address file
record address relative record number file
  See record address file
record format
  for a subfile 9-8
  specifications for externally described files 8-1
record identifying indicators (01-99, H1-H9, L1-L9, LR, U1-U8, RT)
  assigning on input specifications
    for externally described file 12-4
    for program described file 12-4
    rules for 12-3
  general description 12-2
  setting on and off 12-35
  summary 12-33
  with file operations 12-2
record name table, XRCTAB A-4
record sharing
  See file locking by RPG
records, alternate collating sequence table 15-4
records, file translation table 15-5
redirection, file
  definition 6-2
  general description 6-2
relative record number processing 8-14
relative record number record address file
  See record address file
releasing records 6-8
RENAME 6-10
  example 6-12
renaming record format names 6-10
requester
  in INFDS 4-26, 4-31
reserved words
  definition GLOSS-8
  *CANCL 4-47, 5-3
  *DETC
    file exception/error subroutine (INFSR) 4-47
    file information data structure (INFSR) 4-43
    flowchart 5-3
    program exception/errors 4-51
  *DETL
    file exception/error subroutine (INFSR) 4-47
    file information data structure (INFDS) 4-43
    flowchart 5-2
    program exception/errors 4-52
  *FILE 4-42
  *GETIN
    file exception/error subroutine (INFSR) 4-47
    file information data structure (INFDS) 4-38, 4-43
    flowchart 5-2, 5-8

reserved words *(continued)*
*GETIN *(continued)*
program exception/errors 4-52
*IN 12-32
*INIT 4-43, 4-52
*INxx 12-32
*IN,xx 12-32
*OFL
file exception/error subroutine (INFSR) 4-47
file information data structure (INFDS) 4-43
flowchart 5-3
program exception/errors 4-52
*OPCODE 4-42
*PARMS 4-52
*PROGRAM 4-52
*RECORD 4-43
*ROUTINE 4-43, 4-52
*STATUS 4-42, 4-51
*TERM 4-43, 4-52
*TOTC
file exception/error subroutine (INFSR) 4-47
file information data structure (INFDS) 4-43
flowchart 5-3
program exception/errors 4-52
*TOTL
file exception/error subroutine (INFSR) 4-47
file information data structure (INFDS) 4-43
flowchart 5-3
program exception/errors 4-52
resulting indicators (01-99, H1-H9, OA-OG, OV, L1-L9, LR, U1-U8, KQ-KN, KP-KY, RT)
See also individual operation codes
as exception/error indicators 4-40
general description 12-14
rules for assigning 12-14
setting of 12-35
retrieval of data area
implicit 5-2, 10-9
RETRN (return to caller) operation code
program communication 11-11
return points
for file exception/error subroutine
blanks 4-47
*CANCL 4-47
*DETC 4-47
*DETL 4-47
return points *(continued)*
for file exception/error subroutine *(continued)*
*GETIN 4-47
*OFL 4-47
*TOTC 4-47
*TOTL 4-47
for program exception/error subroutine 4-53
return status parameter 6-30
return (RT) indicator
as resulting indicator 12-14
general description 12-21
setting of 12-35
used to end a program 11-11
RETURNCODE data area 3-2
returning from a called program
methods of 11-11
rolldown command key 9-5
rollup command key 9-5
RPG
auto report program phases A-27
compiler phases A-2
default error handler 4-36
processing 2-1
syntax checker 2-3
RPG enhancements
language C-1
RPG logic cycle
detail 5-3
general 1-6, 1-8
RPG restrictions, summary E-1
RPGOBJ 3-7
RT (return) indicator
See return (RT) indicator
run-time array
See also array
definition of 13-2, GLOSS-8
rules for loading 13-3
with consecutive elements 13-4
with scattered elements 13-5
run-time subroutines A-4

S

SAA Flagging 3-10
sample program B-1
SAVDS entry 4-24
search argument
externally described file
description 8-6
referencing a partial key 8-7

AS/400™

SC09-1161-00

Languages: RPG/400 User's Guide

IBM

AS/400™

SC09-1161-00

Languages: RPG/400 User's Guide

First Edition (June 1988)

This edition applies to IBM AS/400™ RPG/400 (Program 5728-RG1), and to any subsequent releases and modifications until otherwise indicated in new editions. Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to:

IBM Canada Ltd.
Information Development
Department 849
1150 Eglinton Avenue East
North York, Ontario, Canada M3C 1H7

IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

AS/400 is a trademark of International Business Machines Corporation, Armonk, NY.

© Copyright International Business Machines Corporation 1988.
IBM is a registered trademark of International Business Machines Corporation, Armonk, NY.

Contents

Chapter 1. An Introduction to RPG/400 and the AS/400 System 1-1
OS/400 1-1
The AS/400 Control Language 1-1
Accessing RPG/400 from the AS/400 System 1-2
Your Choice of Two Environments: AS/400 Native or the System/38 Environment 1-3
The System/38 Environment 1-3
Available Utilities and Languages 1-4
The Source Entry Utility 1-4
The Screen Design Aid 1-4
The Structured Query Language 1-4
Restrictions 1-5
RPG/400 Program Cycle 1-5
General RPG/400 Program Cycle 1-6
Designing Your RPG/400 Program 1-8
Designing the Output 1-8
Designing the Processing 1-9
Designing the Input 1-9
Structured Programming in RPG/400 1-9
Sequential Operation 1-9
Conditional Branching 1-10
Repeating an Operation 1-11
Do While 1-12
Do Until 1-14
Do 1-16
Summary of Structured Programming Operation Codes 1-18
Designing Applications 1-18
Single Program Design 1-18
Modular Program Design 1-19
Examples of Application Design 1-20

Chapter 2. Entering RPG/400 Specifications 2-1
The RPG/400 Specifications 2-1
The Control Specification 2-1
File Description Specifications 2-1
Extension Specifications 2-2
Line Counter Specifications 2-2
Input Specifications 2-2
Calculation Specifications 2-2
Output Specifications 2-2
Entering Your Program 2-3
Using SEU to Enter or Change an RPG/400 Source Program 2-3

Chapter 3. Compiling an RPG/400 Program 3-1
Create RPG/400 Program (CRTRPGPGM) Command 3-2
Using the CRTRPGPGM Command 3-3
Elements of the CRTRPGPGM Command Lines 3-4
Entering Elements from the CRTRPGPGM Command Display 3-4

Entering Only Certain Parameters 3-4
Entering Only the Parameter Values 3-5
CRTRPGPGM Command 3-5
Compiling under the System/38 Environment 3-14

Chapter 4. Error Messages, Testing and Debugging 4-1
Using, Displaying, and Printing Messages 4-1
Using Messages 4-1
Displaying and Printing Messages 4-2
How to Run an RPG/400 Program 4-3
Using a Test Library 4-4
Using Breakpoints 4-6
Example of Using Breakpoints 4-7
Considerations for Using Breakpoints 4-10
Using a Trace 4-10
Example of Using a Trace 4-11
Considerations for Using a Trace 4-12
Using the DEBUG Operation Codes 4-12
Using the RPG/400 Formatted Dump 4-13
File Information Data Structure 4-23
File-Dependent Feedback Information. 4-30
Contents of File Information Data Structure After POST 4-30
Program Status Data Structure 4-33
Exception/Error Handling 4-35
File Exception/Errors 4-38
Exception/Error Indicators 4-39
File Information Data Structure (INFDS) 4-39
Status Codes 4-42
File Exception/Error Subroutine (INFSR) 4-45
Program Exception/Errors 4-50
Program Status Data Structure 4-50
Status Code 4-51
Program Exception/Error Subroutine 4-52

Chapter 5. The Detailed RPG Program Cycle 5-1
Detailed RPG/400 Program Cycle 5-3
RPG Exception/Error Handling Routine 5-8
Match Fields Routine 5-10
Overflow Routine 5-11
Look-Ahead Routine 5-11
Ending a Program without a Primary File 5-11
Program Control of File Processing 5-12

Chapter 6. General File Considerations 6-1
Device Independence/Device Dependence 6-1
Spooling 6-3
Output Spool 6-3
Externally Described/Program Described Files 6-4
Level Checking 6-7
File Locking by RPG/400 6-7
Record Locking by RPG/400 6-8

Unblocking Input Records and Blocking Output Records 6-8
Sharing an Open Data Path 6-9
Using the CL Command RCLRSC 6-9
RPG Specifications for Externally Described Files 6-10
File Description Specifications 6-10
Renaming Record Format Names 6-10
Ignoring Record Formats 6-12
Floating-Point Fields 6-13
Overriding or Adding RPG/400 Functions to the External Description . . 6-13
Output Specifications 6-15
Program Described Files 6-17
Printer Files 6-17
Page Overflow 6-18
Overflow Indicators 6-19
Fetch Overflow Logic 6-21
PRTCTL (Printer Control) Option 6-24
SEQ (Sequential) File 6-27
Special File 6-29
Primary/Secondary Multifile Processing 6-33
Multifile Processing with No Match Fields 6-34
Multifile Processing with Match Fields 6-34
Assigning Match Field Values (M1-M9) 6-35
Processing Matching Records 6-39

Chapter 7. Commitment Control 7-1
Using Commitment Control 7-1
Starting and Ending Commitment Control 7-1
Specifying Files for Commitment Control 7-2
Commitment Control Operations 7-2
Commitment Control Locks 7-3
Commitment Control in the Program Cycle 7-3
Example of Using Commitment Control 7-4

Chapter 8. Using DISK Files 8-1
Externally Described Disk Files 8-1
Record Format Specifications 8-1
Access Path 8-2
Valid Keys for a Record or File 8-6
Valid Search Arguments 8-6
Referring to a Partial Key 8-7
Processing Methods for Externally Described DISK Files 8-8
Program Described Disk Files 8-8
Indexed Files 8-8
Valid Search Arguments 8-8
Sequential Files 8-9
Record Address File 8-12
Limits Records 8-13
Relative Record Numbers 8-14
Externally Described File as Program Described 8-14
Methods for Processing Disk Files 8-14
Relative Record Number Processing 8-14

Consecutive Processing 8-15
Sequential by Key Processing 8-15
Random by Key Processing 8-21
Sequential within Limits Processing 8-22
Keyed Processing Examples 8-22
Valid File Operations 8-29

Chapter 9. Using WORKSTN files 9-1
Intersystem Communications Function (ICF) 9-1
Externally Described WORKSTN Files 9-1
Processing an Externally Described WORKSTN File 9-2
Function Key Indicators 9-4
Command Keys 9-5
Processing WORKSTN Files 9-5
Subfiles 9-7
Use of Subfiles 9-10
Program Described WORKSTN File 9-12
Program Described WORKSTN File with a Format Name 9-12
Output Specifications 9-13
Input Specifications 9-13
Calculation Specifications 9-14
Additional Considerations 9-14
Program Described WORKSTN File without a Format Name 9-14
Input File 9-14
Output File 9-15
Combined File 9-15
Multiple Device Files 9-15
WORKSTN File Examples 9-16
Sample Program 1 – Inquiry 9-17
Sample Program 2 – Data Entry with Master Update 9-23
Sample Program 3 – Maintenance 9-30
Sample Program 4 – WORKSTN Subfile Processing 9-40
Sample Program 5 – Inquiry by Zip Code and Search on Name 9-48
Sample Program 6 – Program Described WORKSTN File within FORMAT Name on Output Specifications 9-57
Sample Program 7 – Variable Start Line 9-59
Sample Program 8 – Read Operation with Time-Out 9-61

Chapter 10. Data Field Formats and Data Structures 10-1
Format of Fields in Files 10-1
Packed Decimal Format 10-1
Zoned Decimal Format 10-3
Binary Format 10-3
Signs 10-5
External Formats 10-5
Internal Format 10-6
Data Structures 10-6
Format of Data Structure Subfields in Storage 10-7
Data Structure Statement Specifications 10-8
Rules for Specifying Data Structure Statements 10-8
Multiple Occurrence Data Structure 10-9

Special Data Structures 10-9
Data Area Data Structure 10-9
File Information Data Structure (INFDS) 10-10
Program Status Data Structure 10-11
Data Structure Subfield Specifications 10-11
Rules for Subfield Specifications 10-12
Data Structure Examples 10-13

Chapter 11. Communicating with Objects in the System 11-1
Calling Other Programs 11-1
CALL (Call a Program) 11-3
PLIST (Identify a Parameter List) and PARM (Identify Parameters) . . . 11-4
Rules for Specifying PLIST 11-4
Rules for Specifying PARM 11-5
OS/400 Graphics Support 11-5
FREE (Deactivate a Program) 11-6
Calling Special Subroutines 11-6
Message-Retrieving Subroutine (SUBR23R3) 11-6
Moving Double-byte Data and Deleting Control Characters (SUBR40R3) 11-8
Moving Double-byte Data and Adding Control Characters (SUBR41R3) 11-10
Returning from a Called Program 11-11
A Normal End 11-11
An Abnormal End 11-12
Return without an End 11-12
Data Areas 11-13

Chapter 12. Using Indicators 12-1
Indicators Defined on RPG/400 Specifications 12-1
Overflow Indicators 12-2
Record Identifying Indicators 12-2
Rules for Assigning Record Identifying Indicators 12-3
Control Level Indicators (L1-L9) 12-4
Rules for Control Level Indicators 12-5
Split Control Field 12-10
Field Indicators 12-13
Rules for Assigning Field Indicators 12-13
Resulting Indicators 12-14
Rules for Assigning Resulting Indicators 12-14
Indicators not Defined on the RPG/400 Specifications 12-15
External Indicators 12-15
Internal Indicators 12-16
First Page Indicator (1P) 12-16
Last Record Indicator (LR) 12-16
Matching Record Indicator (MR) 12-17
Using Indicators 12-17
File Conditioning 12-17
Rules for File Conditioning 12-17
Field Record Relation Indicators 12-18
Assigning Field Record Relation Indicators 12-18
Function Key Indicators 12-21
Return Indicator (RT) 12-21

Halt Indicators (H1-H9) 12-22
Indicators Conditioning Calculations 12-22
Positions 7 and 8 12-22
Positions 9-17 12-23
Indicators Conditioning Output 12-27
Indicators Referred to as Data 12-32
*IN 12-32
*INxx 12-32
Additional Rules 12-32
Summary of Indicators 12-33

Chapter 13. Using Arrays and Tables 13-1
ARRAYS 13-1
Array Name and Index 13-1
The Essential Array Specifications 13-2
Coding a Run-Time Array 13-2
Loading a Run-Time Array 13-3
Array Information in One Record 13-3
Array Information in More Than One Record 13-3
Sequencing Run-Time Arrays 13-4
Coding a Compile-Time Array 13-4
Loading a Compile-Time Array 13-4
Rules for Array Input Records 13-5
Coding a Prerun-Time Array 13-7
Loading a Prerun-Time Array 13-8
Defining More than One Array 13-9
Two Run-Time Arrays 13-9
Mixing Compile-Time and Prerun-Time Arrays 13-9
Arrays in Alternating Format 13-9
Searching Arrays 13-10
Searching an Array without an Index 13-10
Searching an Array with an Index 13-13
Specifying Arrays 13-15
Specifying an Array in Calculations 13-15
Modifying Contents of Arrays 13-16
Adding Entries to Arrays 13-16
Array Output 13-17
Editing Entire Arrays 13-18
Examples of Using Arrays 13-18
TABLES 13-26
LOKUP with One Table 13-27
LOKUP with Two Tables 13-27
Specifying the Table Element Found in a LOKUP Operation 13-30

Chapter 14. Editing Numeric Fields 14-1
Edit Codes 14-1
Simple Edit Codes 14-2
Combination Edit Codes 14-2
User-Defined Edit Codes 14-4
Editing Considerations 14-4
Summary of Edit Codes 14-5

Edit Words 14-8
How to Code an Edit Word 14-9
Parts of an Edit Word 14-9
Forming the Body of an Edit Word 14-10
Forming the Status of an Edit Word 14-13
Formatting the Expansion of an Edit Word 14-14
Summary of Coding Rules for Edit Words 14-14
Formatting Edit Words 14-15
Editing Externally Described Files 14-15

Chapter 15. Alternate Collating Sequence and File Translation 15-1
Changing the Collating Sequence 15-1
Specifying an Alternate Collating Sequence 15-1
Filling Out the Alternate Collating Sequence Coding Sheet 15-1
Formatting the Alternate Collating Sequence Records 15-4
File Translation 15-4
Specifying a File Translation 15-5
Translating One File or All Files 15-5
Translating More than One File 15-6
Specifying the Files 15-6
Specifying the Table 15-6

Chapter 16. Auto Report Feature 16-1
Group Printing 16-1
Specifications 16-1
Examples 16-1
/COPY Statement Specifications 16-4
Modifying Copied Specifications 16-6
Modifying File Description Specifications 16-6
Modifying Input Field Specifications 16-7
Report Format 16-11
Spacing and Skipping 16-11
Placement of Headings and Fields 16-11
Page Headings 16-12
Reformatting *AUTO Page Headings 16-12
Body of the Report 16-13
Overflow of the D/T-*AUTO Print Lines 16-14
Generated Specifications 16-16
Generated Calculations 16-16
Generated Output Specifications 16-17
Programming Aids 16-22
Using CRTRPTPGM to Compile an Auto Report Source Program 16-26
Using the CRTRPTPGM Command 16-27
CRTRPTPGM Command 16-27
Examples of Using Auto Report 16-29
EXAMPLE 1-Sales Report 16-29
EXAMPLE 2-Sales Report with Three Levels of Totals 16-33
EXAMPLE 3-Sales Report with Group Indication 16-35
EXAMPLE 4-Sales Report with Cross-Column Totals 16-39
EXAMPLE 5-Sales Report using copied Specifications 16-42
EXAMPLE 6-Override Copied Input Specifications 16-46

Appendix A. RPG Compiler and Auto Report Program Service Information . A-1
Compiler Overview A-1
Compiler Phases A-2
Major Compiler Data Areas A-4
Compiler Error Message Organization A-4
Run-Time Subroutines A-4
Compiler Debugging Options A-5
*SOURCE Value for the OPTION Parameter A-6
*XREF Value for the OPTION Parameter A-6
*DUMP Value for the OPTION Parameter A-6
*LIST Value for the GENOPT Parameter A-6
*ATR Value for the GENOPT Parameter A-6
*XREF Value for the GENOPT Parameter A-6
*DUMP Value for the GENOPT Parameter A-6
*PATCH Value for the GENOPT Parameter A-6
*OPTIMIZE Value for the GENOPT Parameter A-6
ITDUMP Parameter A-7
SNPDUMP Parameter A-7
CODELIST Parameter A-7
PHSTRC Parameter A-7
Examples of Using Compiler Debugging Options A-7
IRP Layout A-25
Auto Report Program A-27

Appendix B. Sample Program B-1
Sample RPG Program (ORD400) B-1

Appendix C. Differences Between AS/400 RPG/400 and AS/400 RPG II System/36-Compatible C-1
Language Enhancements C-1

Appendix D. Data Communication D-1
Exception/Error Handling with ICF Files D-1
Communications Error Recovery D-2

Appendix E. RPG Restrictions E-1

Appendix F. Distributed Data Management (DDM) Files F-1

Appendix G. Converting from System/38 to AS/400 System G-1
Features of System/38 Not Supported on AS/400 System G-1
Features of AS/400 System That Will Not Run on System/38 . . . . . G-1
Converting to AS/400 System G-1

Appendix H. Glossary of Abbreviations H-1

Appendix I. Using Double Byte Character Set (DBCS) Data in RPG/400 Programs. I-1
Where You Can Use DBCS Data in RPG/400 Programs I-1
How to Work with DBCS Data in RPG/400 Programs I-1
Transparent Literals and Constants I-1

Additional Considerations for Using DBCS Data in RPG/400 Programs . . I-2
Example of Coding DBCS Data in an RPG/400 Program I-2

Glossary of Terms GLOSS-1

Index X-1

About This Manual

Who Should Use this Manual

This manual is a guide for the RPG/400 programming language on the AS/400[1] System. It is intended for people who have a basic understanding of data processing concepts and of the RPG/400 programming language. For a detailed description of RPG/400 specifications and operation codes, see the *Languages: RPG/400 Reference*

The topics this manual covers include:

- Design RPG/400 programs
- Code RPG/400 programs
- Enter and compile RPG/400 programs
- Test and debug RPG/400 programs
- Study coded RPG/400 examples and sample programs.

How This Manual is Organized

This manual is a user's guide to RPG/400. Chapter 1 introduces RPG/400. Chapters 2 to 5 describe coding, entering, compiling, testing, debugging, and putting a program into production. Chapters 6 to 15 describe how to use various elements such as files, indicators, arrays, and tables. The appendixes provide additional information on the System/38 Environment, compiler debugging options, and communications.

What You Should Know

Before you use this manual, you should be familiar with certain information:

- You should know how to describe files to the system and how to create jobs and output queues. This information is contained in *Programming: Data Management Guide*, SC21-9658.
- You should be familiar with your display station (also known as a work station), and its controls. There are also some elements of its display and certain keys on the keyboard that are standard regardless of which software system is currently running at the display station, or which hardware system the display station is hooked up to. Some of these keys are:
  - Cursor movement keys
  - Command keys
  - Field exit keys
  - Insert and delete keys
  - The Error Reset key.

1 AS/400 is a trademark of the International Business Machines Corporation.

This information is contained in *System Operations: Display Station User's Guide* SC21-9744.

- You should know how to operate your display station when it is hooked up to the IBM AS/400 System and running AS/400 software. This means knowing about OS/400 and the Control Language (CL) to do such things as:
- Sign on and sign off the display station
- Interact with displays
- Use Help
- Enter control commands
- Call utilities
- Respond to messages.

To find out more about this operating system and its control language, refer to these IBM publications:

- *Programming: Control Language Reference*, SBOF-0481
- *Programming: Control Language Programmer's Guide*, SC21-8077
- *Programming: Command Reference Summary*, SC21-8076
- *Programming: System Reference Summary*, SC21-8104

- You should know how to call and use certain utilities available on the AS/400 System:
  - The Screen Design Aid (SDA) utility used to design and code displays. This information is contained in *Application Development Tools: Screen Design Aid User's Guide and Reference*, SC09-1171.
  - The Source Entry Utility (SEU), which is a full-screen editor you can use to enter and update your source and procedure members. This information is contained in *Application Development Tools: Source Entry Utility User's Guide and Reference*, SC09-1172.
- You should know how to interpret displayed and printed messages. This information is contained in "Using, Displaying, and Printing Messages" on page 4-1 in Chapter 4, "Error Messages, Testing and Debugging."
- You should be familiar with the RPG/400 program cycle, how indicators affect the program cycle, and how to code entries on the RPG/400 specification sheets.

  These general items about RPG/400 are taught in an RPG/400 coding class. Detailed information on RPG/400 can be found in *Languages: RPG/400 Reference*, SC09-1089.

If You Need More Information

You might need to refer to other AS/400 manuals for specific information about a particular topic. They are listed below:

- *Information Directory*, GC21-9678, which contains a brief description of each manual in the AS/400 library and information on how to order additional publications.
- *Licensed Programs Installation Guide*, SC21-9765, which describes how to install RPG/400 on your system.

- *System Operations: Operator's Guide*, SC21-8082, which describes how to operate the AS/400 System.
- *Programming: Data Description Specifications Reference*, SC21-9620, which describes data description specifications that are used for describing files.
- *Communications: Distributed Data Management User's Guide*, SC21-9600, which contains information about remote communication for the RPG/400 programmer.
- *Programming: Data Base Guide*, SC21-9659, which contains a detailed discussion of the AS/400 data base structure. This manual also describes how to use data description specifications (DDS) keywords.
- *Communications: Programmer's Guide*, SC21-9590, which provides information an application programmer needs to write applications that use AS/400 communications and the Intersystem Communications Function file.
- *Programming: Graphical Data Display Manager Programming Reference*, SC33-0537, and *Programming: Graphical Data Display Manager Programming Guide*, SC33-0536, which provide guidance on the Graphical Data Display Manager (GDDM)[2] for programmers who need to write graphics applications.
- *Programming: Structured Query Language/400 Programmer's Guide*, SC21-9609 and *Programming: Structured Query Language/400 Reference*, SC21-9608 for information on SQL.
- *System/38 Environment Programmer's Guide and Reference*, SC21-9755, which describes migrating from System/38 and converting to AS/400 System.
- *Programming: Data Management Guide*, SC21-9658, which contains information about managing key aspects of the system.

---

2 GDDM is a trademark of International Business Machines Corporation.

AS/400 RPG/400 User's Guide

This guide has 5 parts. Chapter 1 briefly introduces AS/400 RPG/400, including the AS/400 Environment and the System/38 Environment, the RPG logic cycle, and an overview of structured operation codes and application design. Chapters 2-5 describe common tasks in developing RPG/400 programs:

- Coding
- Entering
- Compiling
- Testing
- Debugging.

Chapters 6-10 describe files used for input and output, and data structures. Chapters 11-16 cover miscellaneous elements such as indicators, tables, arrays and the auto report feature. The appendixes contain tables of the more frequently used information in this guide. They also contain additional information which may be useful to programmers migrating from System/38 or simultaneously developing applications for the AS/400 System and System/38.

Note

This manual is designed to guide the programmer in the use of RPG/400, on AS/400. RPG/400 specifications and operations are frequently mentioned. For a detailed description of RPG/400 coding specifications and operation codes see *Languages: RPG/400 Reference*.

Chapter 1. An Introduction to RPG/400 and the AS/400 System

RPG/400 is a programming language designed to make it easier to create business software applications.

RPG is a language under evolution. A slightly different version of RPG is available on each machine that supports it. The AS/400 System is the most recent of these computing systems. You should know that, as well as offering a new enhanced version of RPG, the AS/400 System also supports the previous version of RPG available on System/38 and System/36 computers. For more information see Appendix C, "Differences Between AS/400 RPG/400 and AS/400 RPG II System/36-Compatible" and Appendix G, "Converting from System/38 to AS/400 System."

This chapter provides an overview of the following subjects:

- OS/400 and Control Language
- Accessing RPG/400 from the AS/400 System
- Two environments: AS/400 native environment or the System/38 Environment
- Available languages and utilities
- RPG/400 programming cycle
- Designing your program
- Structured programming in RPG/400
- Designing applications.

OS/400

The operating system that controls all your interactions with the AS/400 System is called OS/400. From your work station, OS/400 allows you to:

- Sign on and sign off
- Interact with the displays
- Use Help
- Enter control commands and procedures
- Respond to messages
- Manage files
- Run utilities and programs.

Refer to the *Information Directory* for a complete list of publications which discuss OS/400.

The AS/400 Control Language

You can manipulate OS/400 with the Control Language (CL). You interact with the system by entering or selecting CL commands. In many cases, the AS/400 System displays a series of CL commands or command parameters appropriate to the situation on the screen. You can then select the desired command or parameters.

The AS/400 CL commands you will be using most often with RPG/400 are:

- STRSEU to call the source entry utility (SEU), a full-screen editor that can be used to enter RPG/400 program code
- CRTRPGPGM to compile RPG/400 source programs
- CRTRPTPGM to compile auto report source programs
- CALL program-name to run an RPG/400 program
- CALL QCL to access the System/38 Environment

The Control Language and all its commands are described in detail in the *Programming: Control Language Reference.*

Accessing RPG/400 from the AS/400 System

When you start working on the AS/400 System, you will see the following screen.

Note: Your site may use a different sign-on display. If it does, contact your system operator for further information.

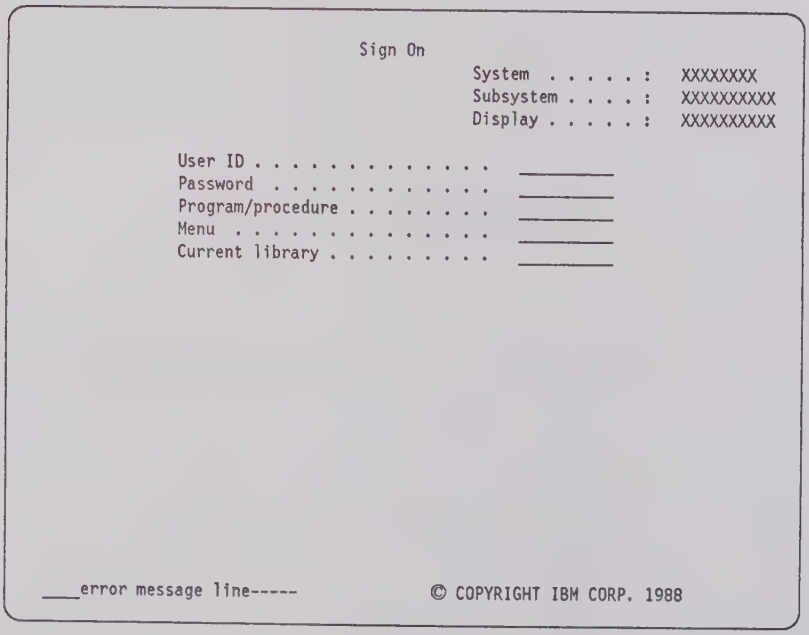


Figure 1-1. The OS/400 Sign-On Display

When you enter your ID and password the following screen appears, and you can begin working in OS/400.

Note: If your implementation is unsecured, the password prompt will not be displayed, and you will not have to enter a password.

```
MAIN                          Main Menu
                                              System: xxxxxxxx
Select one of the following:

      1. User tasks
      2. Office tasks
      3. General system tasks
      4. Files, libraries, and folders
      5. Programming
      6. Communications
      7. Define or change the system
      8. Problem handling
      9. Display a menu

     90. Sign off



Selection or command
 ===>_____________________________________________________________________

F3=Exit      F4=Pompt      F9=Retrieve      F13=User Support
F23=Set initial menu
                                          © COPYRIGHT IBM CORP. 1988
```

Figure 1-2. The OS/400 Main Menu Display

Your Choice of Two Environments: AS/400 Native or the System/38 Environment

The AS/400 System offers increased function over System/38. However, since many RPG/400 programs have been written for the System/38 computer, and because many programmers are already familiar with System/38, the AS/400 System also supports these programs, under the System/38 Environment.

The System/38 Environment

To make your AS/400 emulate a System/38, enter CALL QCL. This command accesses the System/38 Environment. When you are in this environment, you can enter and compile RPG/400 programs, as if you were using a System/38 machine. The file naming conventions are the same as in System/38. You can also enter AS/400 CL commands in the System/38 Environment. You can enter System/38 Environment commands from the native environment by library qualifying commands. The command QSYS38/CRTRPGPGM calls the System/38 Environment RPG III compiler. For more information on the System/38 Environment see the *Programming: System/38 Environment Programmer's Guide and Reference.*

You can use the Source Entry Utility (SEU) to enter your RPG/400 source program interactively. The command you enter to call SEU is the CL command STRSEU. If you specify the TYPE(RPG) parameter on this command, the RPG/400 syntax checker is called and detects RPG/400 syntax errors, statement by statement, while the source

program is entered. Alternatively, you can key a source program on diskettes and upload it into a source file.

> **Note**
>
> This manual describes RPG/400 as it is used on the AS/400 System Native Environment. To find out how to use RPG III in the System/38 Environment, you should refer to Appendix G, "Converting from System/38 to AS/400 System" on page G-1 and to the *System/38 RPG III Reference Manual and Programmer's Guide*, SC21-7725. For information on System/38 devices and commands, refer to the appropriate manuals in the System/38 library.

Available Utilities and Languages

The AS/400 System offers two utilities and a language that you may find useful for programming. They are the Screen Design Aid Utility (SDA), the Source Entry Utility (SEU), and the Structured Query Language (SQL).

The Source Entry Utility

The Source Entry Utility (SEU) is used to enter your code into the system. SEU also provides extensive syntax checking. For more information about SEU, refer to *Application Development Tools: Source Entry Utility User's Guide and Reference.*

The Screen Design Aid

The Screen Design Aid (SDA) is a utility that makes it easier for you to create the displays your program requires. For more information about SDA, refer to *Application Development Tools: Screen Design Aid User's Guide and Reference.*

The Structured Query Language

The AS/400 System allows you to insert Structured Query Language (SQL) statements into RPG/400 programs. SQL statements are entered on a calculation specification. The syntax is shown in Figure 1-3 on page 1-5. The following rules must be observed:

- The starting delimiter /EXEC SQL must be entered into columns 7-15, with the slash in column 7.
- SQL statements can be started on the same line as the starting delimiter.
- SQL statements can be continued on any number of subsequent continuation lines. The continuation line delimiter is the "+" in column 7.
- SQL statements cannot go past column 74.
- The ending delimiter is /END-EXEC and must be entered in columns 7-15 on a separate calculation specification. This signals the end of the SQL statements. Therefore, /END-EXEC must be entered by itself, with no SQL statements following it.

```
C
C     |
C     |
C/EXEC SQL    (the starting delimiter)
C+
C+       (continuation lines containing SQL statements)
C+
.
.
.
C/END-EXEC    (the ending delimiter)
C     |
C     |
C     |
```

Figure 1-3. Syntax for Entering SQL Statements into an RPG/400 Program

A separate command must be entered to process the SQL statements.

Refer to the *Programming: Structured Query Language Programmer's Guide* and the *Programming: Structured Query Language Reference* for the descriptions of how to code SQL statements.

Restrictions

In RPG/400, the only restriction is that SQL statements cannot be specified in the referenced source member of a /COPY statement.

You should not use SQL statements in an RPG auto report program. Instead, you should use the CRTRPTPGM command to process your RPG auto report programs and save the generated RPG/400 source. Auto report will generate RPG/400 source, to which you can add SQL statements. To process your SQL statements and generate an RPG object program, you should use the SQL pre-processor. If SQL statements are processed by the RPG/400 auto report pre-processor, unpredictable results may occur.

Refer to the *Utilities: Source Entry Utility User's Guide and Reference* for information on how the SEU handles SQL statement syntax checking, and to the *Programming: Structured Query Language Programmer's Guide* and the *Programming: Structured Query Language Reference* for more information on the SQL pre-processor.

RPG/400 Program Cycle

Part of the logic for an RPG/400 program is supplied by the compiler. The logic the compiler supplies is called the *program cycle*. The program cycle is a series of steps that the program goes through in a specific order for each record that is read.

The information that you code on RPG/400 specifications in your source program need not specify when records should be read or written. The RPG/400 compiler can supply the logical order for these operations when your source program is compiled.

Depending on the specifications you code, your program may or may not use each step in the cycle.

A primary file (identified by a P in position 16 of the file description specifications) indicates to the compiler that the program cycle is used to control input operations for the primary and all secondary files. A full procedural file (identified by an F in position 16 of the file description specifications) indicates to the compiler that input is controlled by program-specified calculation operations (for example, READ and CHAIN).

A program can consist of:

- One primary file and one or more secondary files
- Only full procedural files
- A combination of one primary file and one or more full procedural files in which some of the input is controlled by the cycle, and other input is controlled by the program
- No files (for example, input can come from a parameter list or a data area data structure).

General RPG/400 Program Cycle

Figure 1-4 on page 1-7 shows the specific steps in the general flow of the RPG/400 program cycle. A program cycle begins with step 1 and continues through step 7, then begins again with step 1.

The first and last time a program goes through the RPG/400 cycle differ somewhat from the normal cycle. Before the first record is read the first time through the cycle, the program resolves any parameters passed to it, writes the records conditioned by the 1P (first page) indicator, and processes any heading or detail output operations having no conditioning indicators or all negative conditioning indicators. For example, heading lines printed before the first record is read might consist of constant or page heading information or fields for reserved words, such as PAGE and UDATE. In addition, the program bypasses total calculations and total output steps on the first cycle.

During the last time a program goes through the cycle, when no more records are available, the LR (last record) indicator and L1 through L9 (control level) indicators are set on. The program processes the total calculations and total output, and the program ends.

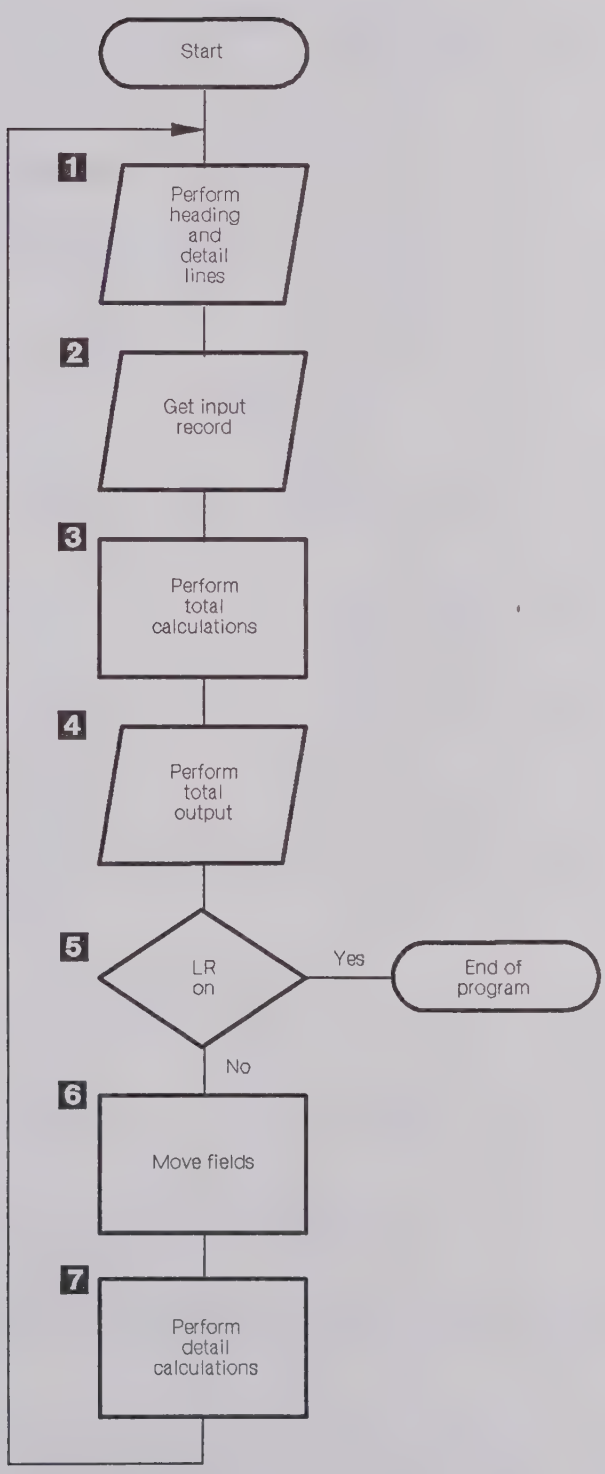


Figure 1-4. RPG Program Logic Cycle

1. RPG/400 processes all heading and detail lines (H or D in position 15 of the output specifications).
2. RPG/400 reads the next input record and sets on the record identifying and control level indicators.
3. RPG/400 processes total calculations, which are conditioned by an L1 through L9 or LR indicator, or an L0 entry.
4. RPG/400 processes all total output lines (identified by a T in position 15 of the output specifications).
5. RPG/400 determines if the LR indicator is on. If it is on, the program is ended.
6. The fields of the selected input records are moved from the record to a processing area. RPG/400 sets on field indicators.
7. RPG/400 processes all detail calculations (those not conditioned by control level indicators in positions 7 and 8 of the calculation specifications) on the data from the record read at the beginning of the cycle.

Designing Your RPG/400 Program

Designing a program includes:

- Deciding what output you need from your program
- Deciding what processing will produce the output you need
- Deciding what input is required by and available to your program.

This sequence may seem backwards because it starts at the results (the output) and ends at the beginning (the input). Designing the output first is like knowing where you are going before you set out on a trip: it helps you decide the best way to get there.

Designing the Output

Your program will produce output records. You must decide what you will do with those records. In general, you have three choices (or any combination of the three choices):

- You can display them.
- You can print them.
- You can store them.

If you want to display the output records at your display station, you have to decide what information you want displayed and how you want it laid out. To define how you want your displays laid out, you use the display layout sheet. Then you can use the Screen Design Aid (SDA) utility to create your display screens. For more information about SDA, refer to *Application Development Tools: Screen Design Aid User's Guide and Reference.*

If you want to print the output records, you have to decide what information you want printed (which fields from which records) and how you want it laid out on the printed report. To indicate how you want the printed report laid out, use the printer layout sheet.

If you want to keep the output records in storage, you have to decide what information you want to keep and how you want to organize the fields in the output records.

After you design all your output records, you code those records on the RPG/400 file description specifications and output specifications.

Designing the Processing

Designing the processing means planning the calculations that will produce the necessary output. When you design the processing, you must be aware of how the RPG/400 program cycle works. The RPG/400 program cycle controls certain read/write operations done on each record, so the program cycle partly determines how you can process your data. The program cycle is described in more detail under "RPG/400 Program Cycle" on page 1-5.

Designing the Input

After you decide what output you need and what calculations will produce that output, the next step is to determine where the input data for your program will come from. It might come from one or more files already on the system, from one or more display stations on your system, from one or more other systems, or from a combination of these sources. You have to know the names used for input files, the location of fields in the input records, the sequence of record types, the formats of numeric data, and the indicators used. When you know this information, you can describe your input records on the RPG/400 input specifications.

Structured Programming in RPG/400

Structured programming is an approach to design, and coding that makes programs easy to understand, debug, and modify.

Three structures used in every computer program are:

- Sequential operation
- Conditional branching
- Repeating an operation based on a certain condition.

Ideally, a structured program is a hierarchy of modules that can have a single entry point and a single exit point. Control is passed downward through the structure without unconditional branches to higher levels of the structure.

This section discusses how these structures can be implemented in RPG/400.

Sequential Operation

Sequential operation means any series of instructions that are processed one after another, rather than transferring control to some other part of the program.

Conditional Branching

An example of an If-Then-Else conditional branching structure in simple English is:

IF the weather is cold,

THEN I will wear my coat;

ELSE, I will leave my coat at home.

Figure 1-5 is a flowchart of a conditional branch.

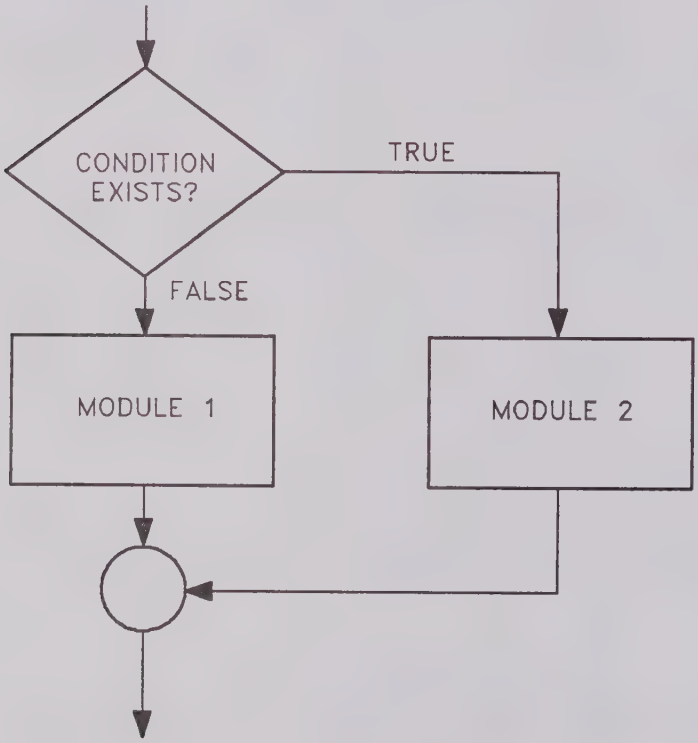


Figure 1-5. Flowchart of a Conditional Branch

In RPG/400, the If-Then-Else structure is implemented through the operation codes IFxx, ELSE, and END. Figure 1-6 shows a design for a conditional branch using the IFxx, ELSE, and END operations.

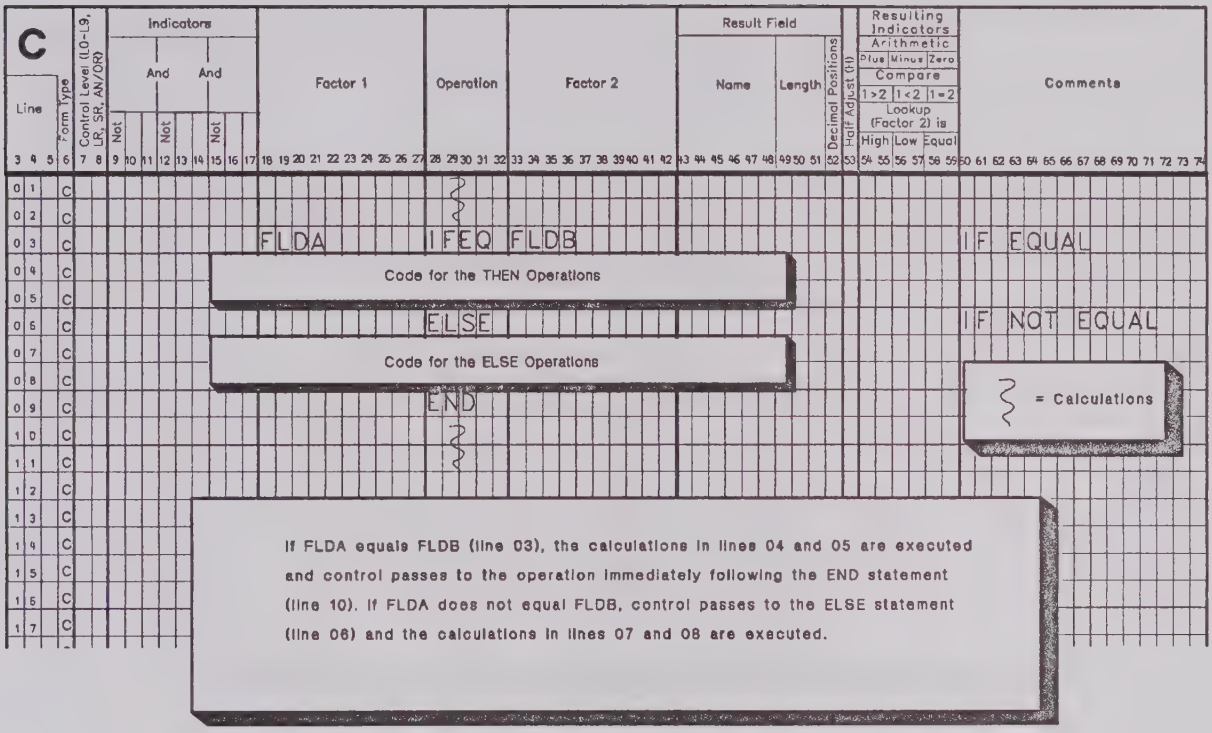


Figure 1-6. Design for a Conditional Branch Using the IF/ELSE/END Operations

There are three other ways to create conditional branches:

- The CASxx operation
- The GOTO operation and conditioning indicators
- The CABxx operation.

You can also create a branch to a subroutine with the EXSR operation and conditioning indicators.

Repeating an Operation

RPG/400 implements three repeat operations – Do While, Do Until and Do – by means of the DOWxx, DOUxx and DO operation codes and the END operation code.

Do While

If you test the condition first and then process the operation(s), the structure is called a Do While. An example of a Do While is:

1. Compare a sum with 5.
2. If the sum is less than 5, add 1 to the sum.
3. Repeat steps 1 and 2 until the sum is equal to or greater than 5.

Figure 1-7 is a flowchart of a Do While, and Figure 1-8 on page 1-13 illustrates the coding of a Do While.

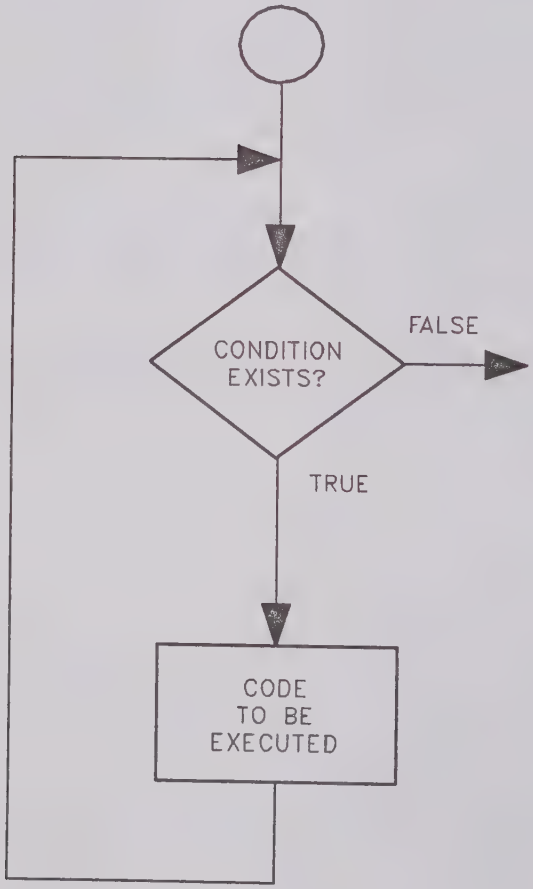


Figure 1-7. Flowchart of a Do While

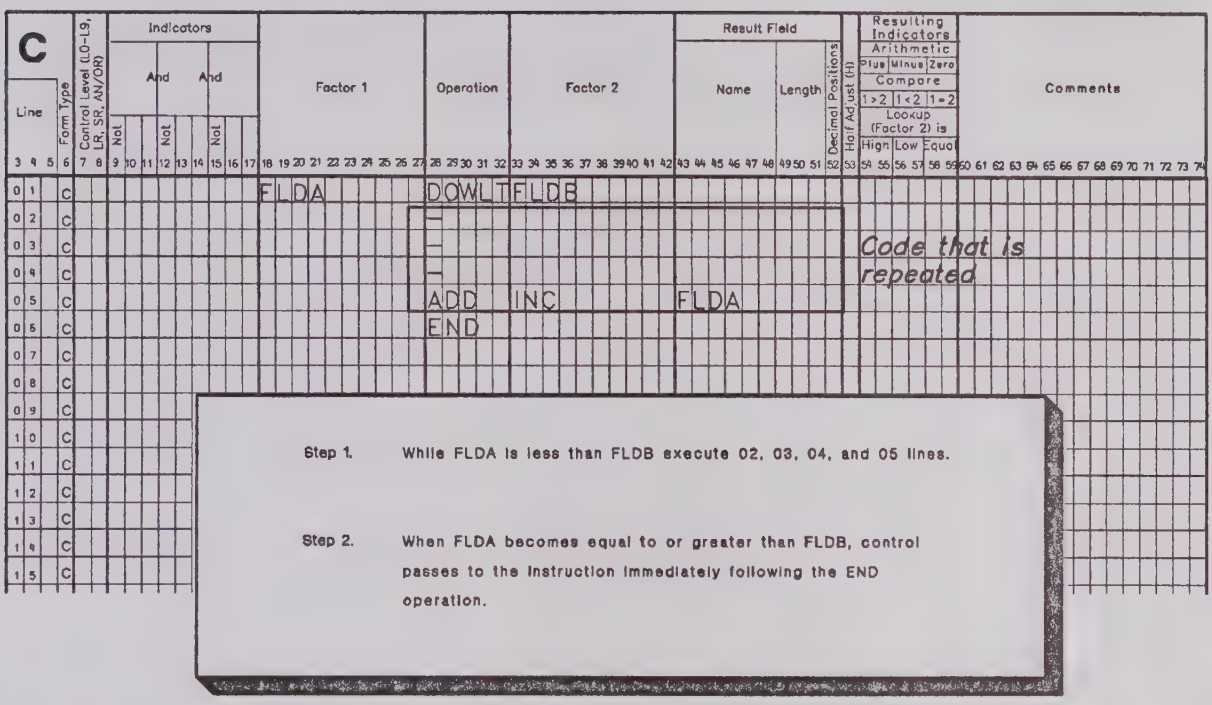


Figure 1-8. Design for a Do While Using the DOWxx Operation Code

Notice in Figure 1-8 (the Do While) that the program first tests if the condition is true (line 01). If it is true, the code between the DOW and the END operations is processed. Then the program goes back to line 1 to test again whether the condition is still true, and the entire cycle is repeated. If the condition is no longer true, control passes to the instruction immediately following the END operation.

Do Until

If you process the operation(s) first and then test the condition, the structure is called a Do Until. An example of a Do Until is:

1. Add 1 to a sum.
2. Compare the sum with 5.
3. Repeat steps 1 and 2 if the sum is less than 5.

Figure 1-9 is a flowchart of a Do Until, and Figure 1-10 on page 1-15 illustrates the coding.

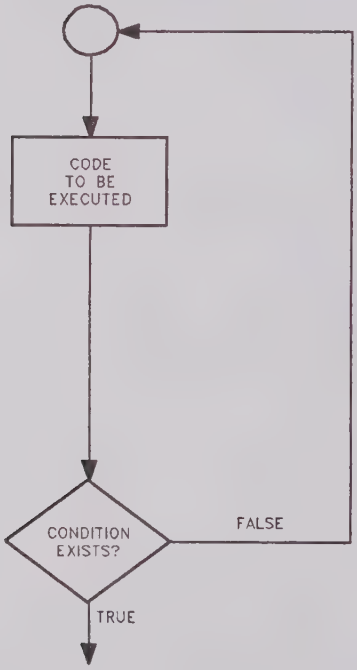


Figure 1-9. Flowchart of a Do Until

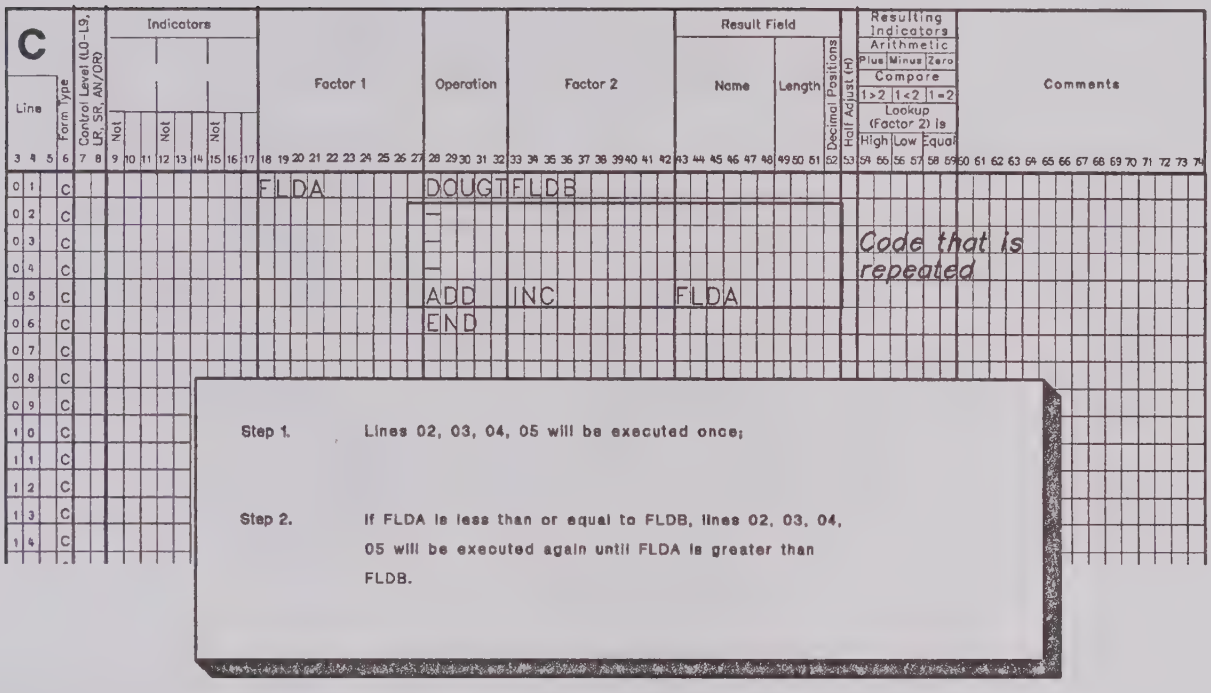


Figure 1-10. Design for a Do Until using the DOUxx Operation Code

Notice in Figure 1-10 (the Do Until) that the program first processes the operations on lines 02 through 05, and then tests (line 01) to see if the condition is met. If this condition is not met, lines 02 through 05 are processed again. The program continues looping until the condition becomes true. Then control passes to the line immediately following the END operation.

Do

In its simplest form, the Do operation using the DO and the END operation codes is coded in the RPG/400 calculation specifications as follows:

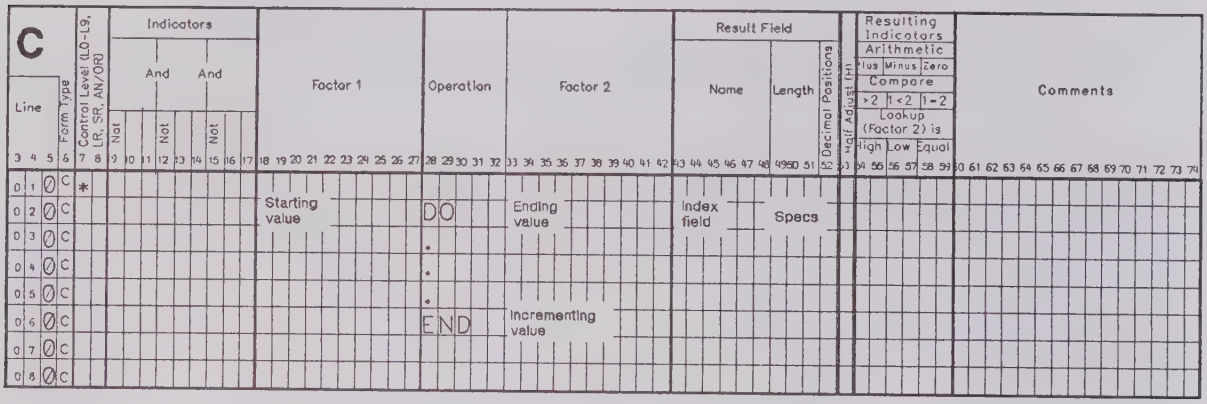


Figure 1-11. Coding of a Do Operation onto the Calculation Specifications

This is how the Do operation works:

1. Set index field (result field) to starting value (factor 1).
2. Test if the index field value is greater than the ending value (factor 2).

   If the index field value is greater than the ending value, control passes to the statement following the END statement.
3. If the index field value is not greater than the ending value, the operations between the DO statement and the END statement are processed.
4. At END, the index field value is incremented by the increment value specified in factor 2 on the END statement, or by 1 if the increment is not specified.
5. Control passes to Step 2 above.

Figure 1-12 on page 1-17 is a flowchart of a Do operation, and Figure 1-13 on page 1-17 illustrates the coding.

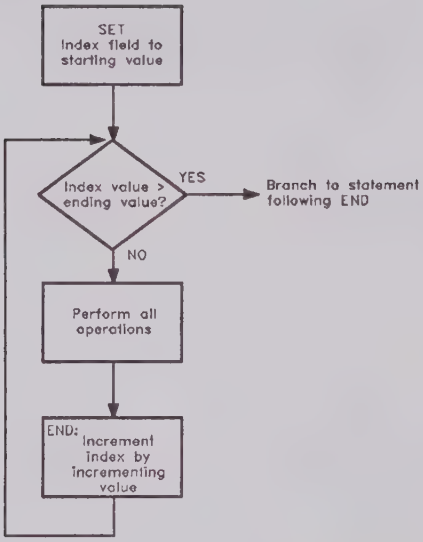


Figure 1-12. Flowchart of a Do Operation

C

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: Not	Not	Not	Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic: Plus	Minus	Zero	Compare: >2	1<2	1=2	Lookup (Factor 2) is: High	Low	Equal	Comments
01	C						1	DO	101	INDEX		30											
02	C						INDEX	ADD	TOTAL	TOTAL		50											
03	C							END	2														
04	C																						
05	C																						
06	C																						
07	C																						
08	C																						
09	C																						
10	C																						
11	C																						
12	C																						
13	C																						
14	C																						

Step 1	Index is set equal to 1. (INDEX is a three-digit integer field).
Step 2	A test is made to determine if INDEX is greater than 101. If yes, the statement after the END is executed.
Step 3	If INDEX is not greater than 101, ADD INDEX (1, 3, 5...) to TOTAL.
Step 4	Increment INDEX by 2.
Step 5	Go to Step 2 above.

Figure 1-13. Design for a Do Operation Using the DO and END Operation Codes

Summary of Structured Programming Operation Codes

The structured programming operation codes are:

- IFxx (If/Then)
- ELSE (Else Do)
- END (End)
- DO (Do)
- DOWxx (Do While)
- DOUxx (Do Until).
- CASxx (Case)

where xx can be:

GT	Factor 1 is greater than factor 2.
LT	Factor 1 is less than factor 2.
EQ	Factor 1 is equal to factor 2.
NE	Factor 1 is not equal to factor 2.
GE	Factor 1 is greater than or equal to factor 2.
LE	Factor 1 is less than or equal to factor 2.
Blanks	Factor 1 is not compared to factor 2 (unconditional processing). This is valid for CASxx operation only.

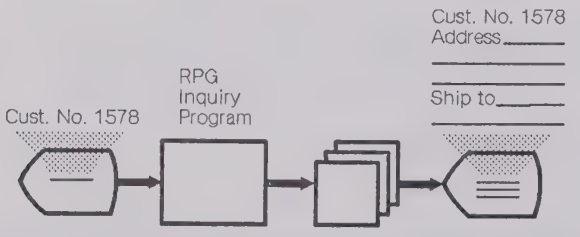
Designing Applications

Application design involves determining whether to create one program to do all of the required functions or to create multiple programs to make up an "application".

Single Program Design

In a single program design, all functions are done within one program. Single program design applies to both batch and interactive programs, and is best used when there are few, relatively simple functions.

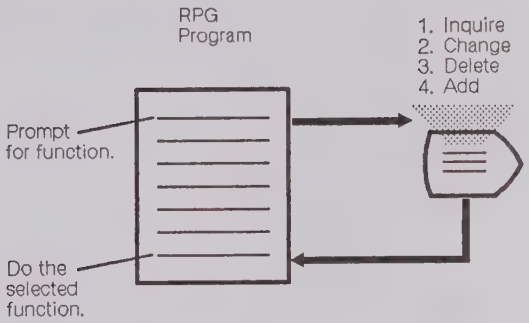
For example, an interactive inquiry program that accepts a customer number from an operator, finds the corresponding record in a customer master file, and then displays the record is a simple program that could have a single program design.



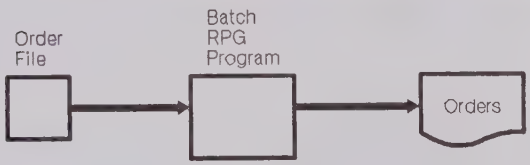
A slightly more complex program that might also have a single program design is a file maintenance program that allows an operator to:

- Inquire into a record
- Change a record

- Delete a record
- Add a record.



An example of a batch program that has a single program design is a program that prints a list of orders that each operator has entered during the day.



Modular Program Design

Modular program design involves using multiple programs to do multiple functions, one function per program. Modular program design can be applied to both batch programs and interactive programs. For example, the order entry application shown in Figure 1-14 is designed to have four programs:

- An RPG/400 or CL mainline program
- An RPG/400 program that prompts for the customer number and shows customer information on the display
- An RPG/400 program that accepts input of line items from the order
- An RPG/400 program that calculates totals for the order.

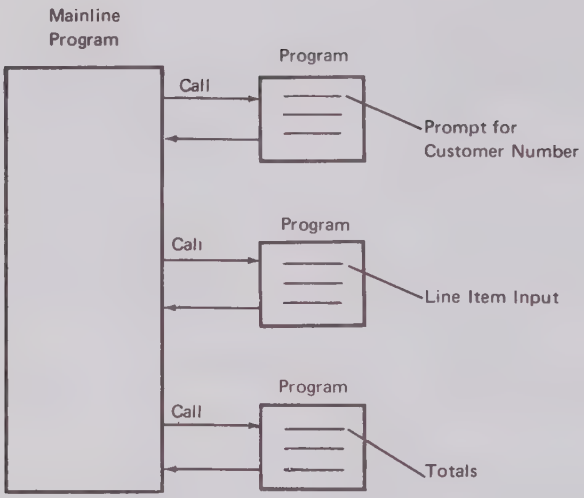


Figure 1-14. Modular Design for an Order Entry Application

A modular program design has several potential advantages:

- Designing, coding, testing, and maintaining several small programs can be easier than designing, coding, testing, and maintaining one large, complex program. Often, this choice is a matter of your personal preference, but it is often beneficial to keep your programs small and as simple as possible.
- CL functions can be requested from RPG/400 programs because the AS/400 System allows RPG/400 programs and CL programs to call one another.

A long-running single program might have sections of code that run infrequently. A modular design could arrange to have the seldom-used code called only when needed.

A potential disadvantage of modular program design is the additional linking of programs that is required. These links take time to code and might require additional system overhead for program processing.

Examples of Application Design

Following are descriptions of modular programs that illustrate some design approaches.

The order entry function shown in Figure 1-15 has three subfunctions:

- Accept heading information about an order
- Accept line item input from the order
- Calculate totals for the order.

One way to design this application is to have a CL mainline program call RPG/400 programs to do the functions.

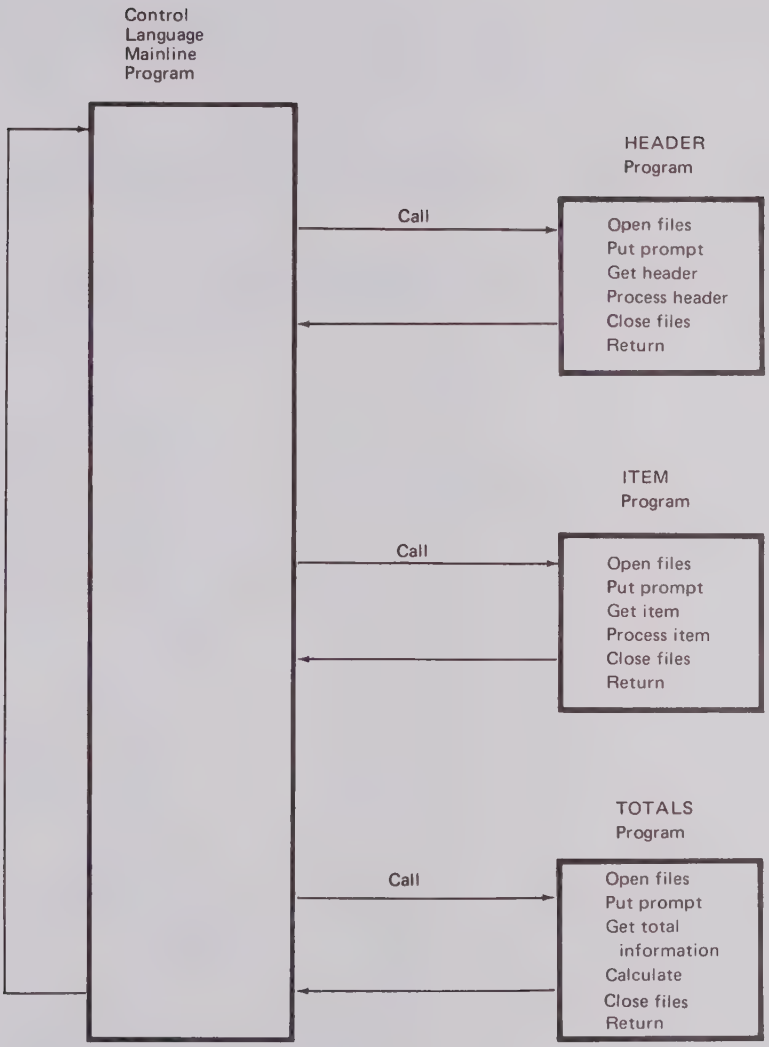


Figure 1-15. Example of Application Design for an Order Entry Function

Each of the RPG/400 programs:

- Opens files
- Displays a prompt for user information and input
- Accepts input from the user
- Processes the information
- Closes the files
- Returns to the mainline program.

Notice that the following events occur after a user enters a display:

1. The input is processed.
2. Files are closed.
3. Control returns to the mainline program.

4. The mainline program calls the next program.
5. That program prompts for user input.

All input and output from work stations and all opening and closing of files occurs in the RPG/400 programs. Therefore, the user might have to wait after entering a display before seeing the next display.

A modification of the previous design that might shorten response times and make more efficient use of system resources is shown in Figure 1-16:

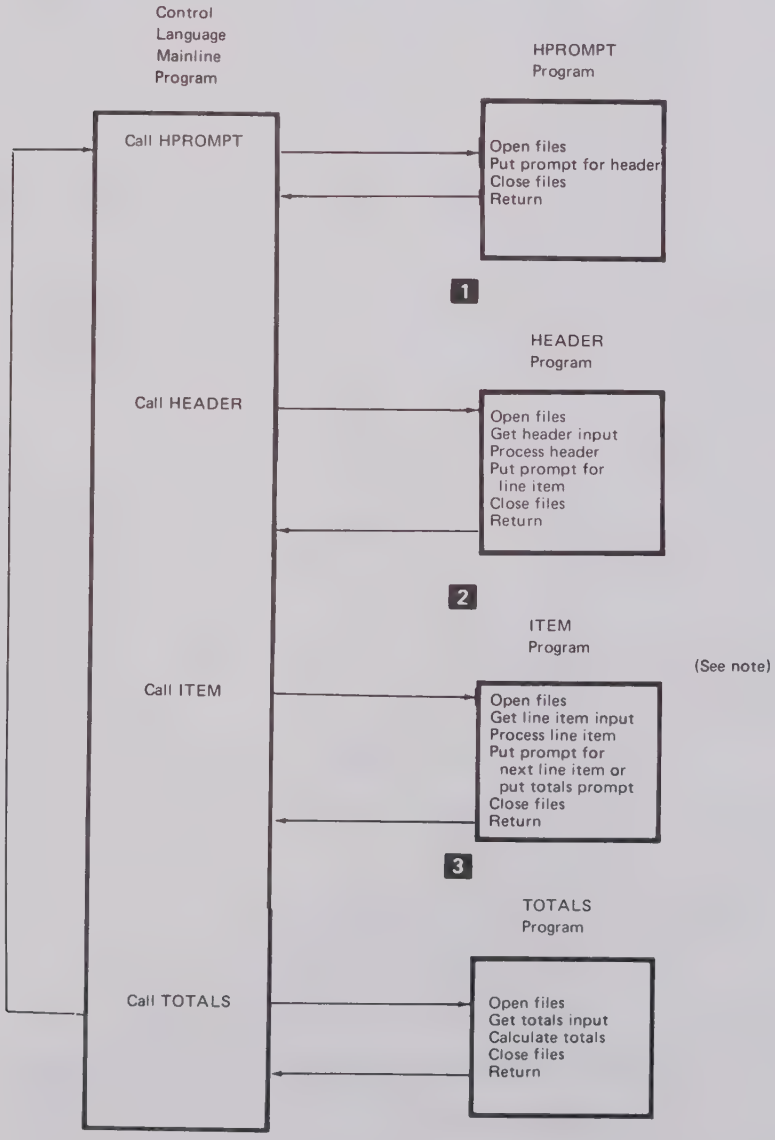


Figure 1-16. Example of Modified Application Design for an Order Entry Function

Note: Rather than returning unconditionally to the mainline program, the ITEM program could be designed to loop within itself as long as line items are being entered.

This modification allows user data entry to occur while programs are started and files are opened and closed. The overlap of data entry and AS/400 processing occurs at points **1**, **2**, and **3**.

For the previous two examples of modular program design, all input from and output to work stations occurs in the programs. For the example in Figure 1-17, these operations occur in an RPG/400 mainline program.

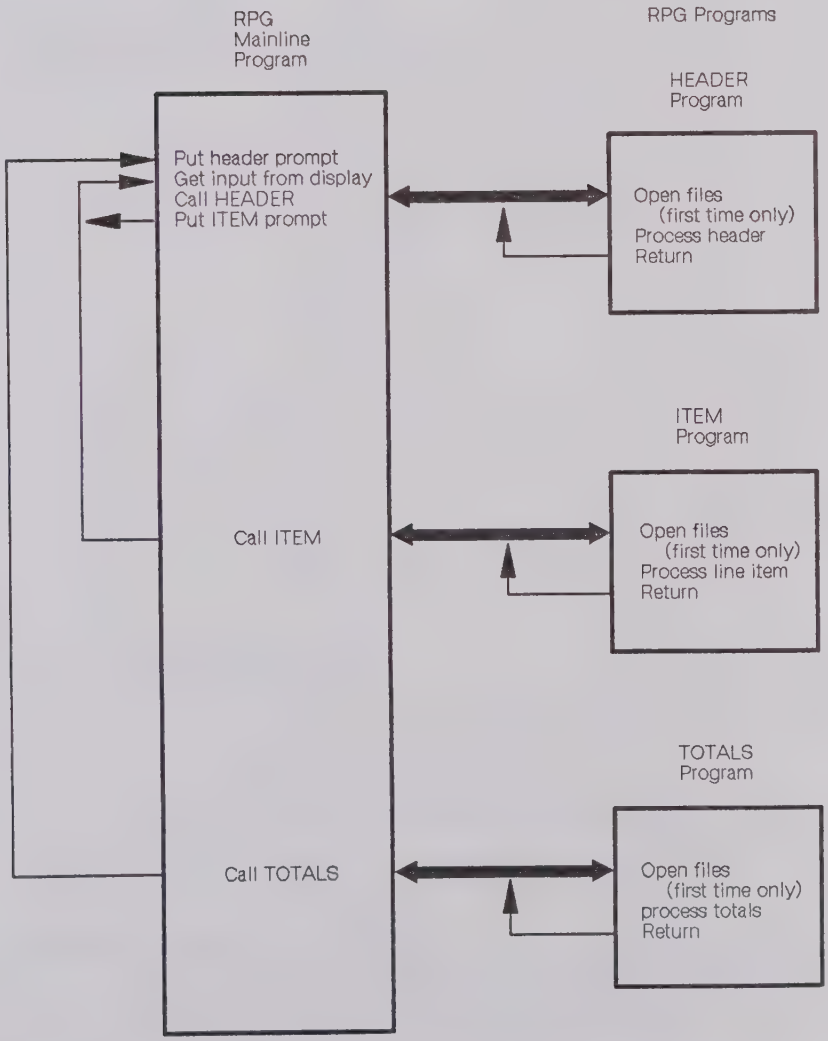


Figure 1-17. Example of Application Design with Input and Output in Mainline Program

The input from the display determines which program to call. If a header is read, HEADER is called and the header record is passed as a parameter. If a line item is read, ITEM is called and a line item record is passed as a parameter. If total information is read, TOTALS is called and a total record is passed as a parameter.

The programs leave files open until the job ends, which eliminates open and close processing time for these files. The programs do not end when they return to the mainline program.

Chapter 2. Entering RPG/400 Specifications

After designing your program, you must write the individual statements that you will combine into a source program. These statements should be coded on RPG/400 specification sheets. Each line coded on a specification sheet represents a statement in the source program. Each specification sheet contains 80 columns. Column headings indicate the kind of information to code in particular columns.

You can also use a text editor, such as SEU, to enter this information directly into the system and thus begin creating your source program on-line.

The RPG/400 Specifications

There are seven kinds of specifications:

- Control specifications
- File description specifications
- Extension specifications
- Line counter specifications
- Input specifications
- Calculation specifications
- Output specifications.

Each of these specifications is described briefly in this chapter. For detailed descriptions see the *Languages: RPG/400 Reference.*

RPG/400 programs do not have to use all specifications. A typical program may use file description, input, calculation, and output specifications.

When your source program is compiled, the RPG/400 specifications must be in the correct sequence, as specified above.

The Control Specification

The control specification provides the RPG/400 compiler with information about your program and your system. This includes:

- Name of the program
- Date format for the program
- Whether an alternative collating sequence or file translation is used.

Note: The control specification is optional.

File Description Specifications

File description specifications describe all the files that your program uses. The information for each file includes:

- Name of the file
- How the file is used
- Size of records in the file
- Input or output device used for the file

- Whether the file is conditioned by an external indicator.

Extension Specifications

Extension specifications describe all record address files, table files, and array files used in the program. The information includes:

- Name of the file, array, or table
- Number of entries in a table or array input record
- Number of entries in a table or array
- Length of the table or array entry.

Line Counter Specifications

Line counter specifications describe the page or form on which output is printed. The information includes:

- Number of lines per page
- Line of the page at which overflow occurs.

Input Specifications

Input specifications describe the records and fields in the input files used by the program. The information for each record includes:

- Name of the file
- Sequence of record types
- Whether record-identifying indicators, control-level indicators, field-record-relation indicators, or field indicators are used
- Whether data structures, look-ahead fields, record identification codes, or match fields are used
- Type of each field (alphanumeric or numeric; packed-decimal, zoned-decimal, or binary format)
- Location of each field in the record
- Name of each field in the record.

Calculation Specifications

Calculation specifications describe the calculations to be done on the data and the order of the calculations. Calculation specifications can also be used to control certain input and output operations. The information includes:

- Control-level and conditioning indicators for the operation specified
- Fields or constants to be used in the operation
- The operation to be processed
- Whether resulting indicators are set after the operation is processed.

Output Specifications

Output specifications describe the records and fields in the output files and the conditions under which output operations are processed. The information includes:

- Name of the file
- Type of record to be written
- Spacing and skipping instructions for PRINTER files
- Output indicators that condition when the record is to be written
- Name of each field in the output record

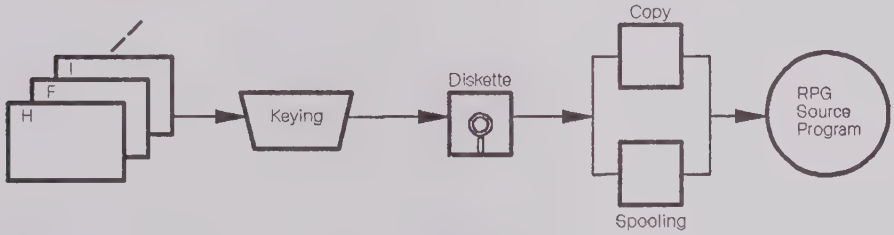
- Location of each field in the output record
- Edit codes and edit words
- Constants to be written
- Format name for a WORKSTN file.

Entering Your Program

After you have written your RPG/400 program on the specifications forms, you must enter it into source files in the system. You can enter the source program in two ways:

- Interactively by using the Source Entry Utility (SEU):

- In a batch manner (that is, from diskette) by using either the OS/400 copy or spooling functions:



See the *Data Management Guide* for more information on how to use the copy or spooling function for batch entry of the source program.

Note: Whichever method of source entry you use, lowercase letters can only be used in literals, constants, comments, array data, and table data. All other information must be in uppercase letters.

Using SEU to Enter or Change an RPG/400 Source Program

Enter the command STRSEU to call SEU. The type you must enter to use the syntax checker is RPG.

The RPG/400 syntax checker checks the specification line for errors as you enter them. (This syntax checker cannot be called from RPG/400.) The RPG/400 syntax checker checks each position of the specification line for valid entries; it checks that all field, indicator, and operation code names are valid; it checks that the proper fields are specified for each operation code (for example, the arithmetic operations must have a result field entry); and it checks that literals are specified correctly. If a position contains an invalid entry, an error message is displayed and you can correct the error at this time.

The syntax checker does not check the compiler directive /TITLE, the comment statements, or the records entered with 80-column record format. In addition, the syntax checker cannot detect logic or relational errors between two or more statements (for example, if no TAG exists for a GOTO operation). These errors are detected by the RPG/400 compiler when you compile your program.

For a complete description of how to enter or update an RPG/400 source program using SEU, see the *Utilities: Source Entry Utility User's Guide and Reference.*

Chapter 3. Compiling an RPG/400 Program

There are two environments that you can compile source programs from: AS/400 Native Environment, and the System/38 Environment. Consequently, there are two ways of compiling source programs and this chapter describes each one:

- Using the CL command CRTRPGPGM to compile an RPG/400 source program in AS/400 Native Environment.
- Using the CL command CRTRPGPGM to compile an RPG/400 source program in the System/38 Environment.

This chapter also contains information on interpreting a compiler listing.

To compile a program you must ensure that the library QTEMP is in the library list. The CL command CRTRPGPGM calls the compiler to create an RPG/400 program object and a listing. (If externally described files are used in the program, OS/400 provides information about the files to the program during compilation.) The following figure shows an overview of the compilation process:

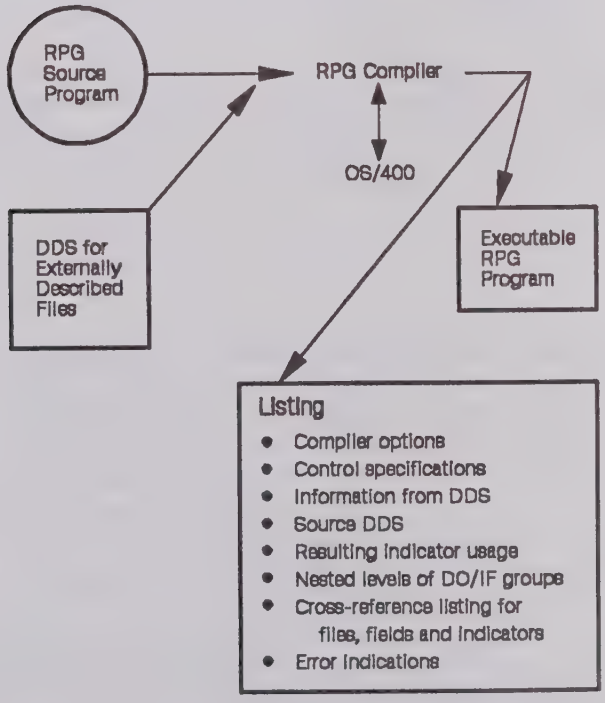


Figure 3-1. Overview of the Compilation Process

The compiler checks the syntax of the RPG/400 source program line by line and the interrelationships between the lines. For example, it checks that all field names are defined and, if a field is multiply defined, that each definition has the same attributes.

The RPG/400 compiler supports a source file record length of 102. In addition to the usual fields of sequence number (6 characters), last modified date (6 characters), and the data (80 characters), a field of 10 characters that can contain additional information is placed at the end of the record (positions 93-102). This information is not used by the RPG/400 compiler but is placed on the extreme right of the compiler listing. You, the programmer, place information into this field. If you want to use the additional field, create a source file with a record length of 102. The AS/400 System has an IBM-supplied RPG/400 source file called QRPGSRC, which has a record length of 92.

Create RPG III Program (CRTRPGPGM) Command

To compile an RPG/400 source program into a program object, you must enter the CL command CRTRPGPGM (Create RPG III Program) to call the RPG/400 compiler. RPG/400 program objects are created with the public authority of *CHANGE. You may want to change this authority to maintain greater security on your system.

If the RPG/400 compiler stops because of errors, the escape message QRG9001 is issued. A CL program can monitor for this exception by using the CL command MONMSG (Monitor Message). See Chapter 4, "Error Messages, Testing and Debugging."

The compiler creates and updates a data area with the status of the last compile. This data area is named RETURNCODE, is 400 characters long, and is placed into library QTEMP. You can access the RETURNCODE data area by specifying RETURNCODE in factor 2 of an *NAMVAR DEFN statement. The data area RETURNCODE has the following format:

Byte	Content and Meaning
1	Character '1' means program was created
2	Character '1' means compile failed because of compiler errors
3	Character '1' means compile failed because of source errors
4	Character '1' means compiled from source generated by auto report
5	Character '1' means program resolution monitor was not called because *NOGEN option was selected on CRTRPGPGM command
6-10	Number of source statements
11-12	Severity level from command
13-14	Highest severity on message diagnostic
15-20	Number of errors found in program
21-26	Compile date
27-32	Compile time
33-100	Not set

Byte	Content and Meaning
101-110	Program name
111-120	Program library name
121-130	Source file name
131-140	Source file library name
141-150	Source file member name
151-160	Compiler listing file name
161-170	Compiler listing library name
171-180	Compiler listing member name
181-190	Auto report source file name
191-200	Auto report library name
201-210	Auto report member name
211-370	Not set
371-378	Size of intermediate representation of program passed to program resolution monitor
379	Not set
380-384	Total compile time
385	Not set
386-390	Time used by compiler
391-395	Time used by program resolution monitor
396-400	Time used by translator

All object names specified on the CRTRPGPGM command must be composed of alphanumeric characters, the first of which must be alphabetic. The full AS/400 naming convention is allowed. The length of the names cannot exceed ten characters. See the *Programming: Control Language Reference* for a detailed description of OS/400 object naming rules and for a complete description of OS/400 command syntax.

It is very unlikely that the system internal size limits for a program will be exceeded. However, if these limits are exceeded, the program must be rewritten, usually as multiple programs.

Using the CRTRPGPGM Command

You can call the CRTRPGPGM compiler in one of three ways:

- Interactively from the CRTRPGPGM command display screen using prompts. You invoke the display, illustrated in Figure 3-2 on page 3-6 and Figure 3-3 on page 3-10, by typing the CL command CRTRPGPGM and then pressing F4.

- Entering CRTRPGPGM followed by only those parameters by keyword that override the default settings. This statement is entered on the command line interactively, or may be done as part of a batch input stream.
- Entering CRTRPGPGM followed only by the parameter values, in the proper sequence. This method is most often used when you are submitting the compile request as part of a batch input stream, or if you are including the compile request as part of a CL program. This method may also be used interactively but you are limited by CL to entering only the first three parameter values.

Note: Any default on the CRTRPGPGM command or any other CL command can be changed using the CL command CHGCMDDFT (Change Command Default). Refer to the *Programming: Control Language Reference* for more information.

Elements of the CRTRPGPGM Command Lines

The descriptions that follow refer to the three elements of the compiler command line:

- The CL compiler command word CRTRPGPGM
- The parameter, which is referred to by a keyword such as PGM, SRCFILE, GENOPT, and so on.
- The value for the parameter. This can be a predefined value or an object name.

All object names must consist of alphanumeric characters. The first character must always be alphabetic, and the length of the name cannot exceed ten characters. The full AS/400 naming convention is allowed.

Entering Elements from the CRTRPGPGM Command Display

Type CRTRPGPGM then press F4. This causes the CRTRPGPGM prompt screens to appear. Press F10 to get additional parameters. Those screens, and the values you can enter on them, are documented later in this chapter.

Each parameter on the screen displays a default value. Move the cursor past items where you want the default value to apply. Type over any items where you want to set a different value or option. If you are not sure about what to set a particular parameter to, type a question mark "?" as the first character in that field and press Enter to receive more detailed information. The "?" must be followed by a blank.

When you have set all values to your satisfaction, press Enter.

Entering Only Certain Parameters

All of the CRTRPGPGM parameters have default values. Simply type CRTRPGPGM, followed by only those parameters (specified by keyword) whose default settings you want to override. Separate parameters by spaces; enter values for each parameter by enclosing the value or values in parentheses.

For example, to change the program and library name, and accept default values for all other parameters, enter:

```
CRTRPGPGM PGM(newlibrary/newname)
```

Entering Only the Parameter Values

You have the choice of entering only the parameter values without specifying the parameter keywords. Since, in this case, there is no keyword to tell the system which value belongs to which parameter, you must enter all the values in the sequence shown below. You need not enter the entire set of options, but you must enter the options for all the parameters up to the one you are interested in. The system will use the default values for the remaining parameters.

For example, to compile a source program in file QRPGSRC in member ABC in library SRCLIB enter:

```
CRTRPGPGM QTEMP/ABC SRCLIB/QRPGSRC *PGM
```

Notice that you also had to enter names for the program and library for the compiled program. The system recognizes which option belongs to which parameter by the position of the value on the compiler command line. You can enter a maximum of three parameter values positionally.

For more information see *Programming: Control Language Reference.* This information applies only to native commands. For System/38 Environment commands, look in the appendixes of this book and System/38 manuals.

CRTRPGPGM Command

Following are examples of the prompt screens for the CRTRPGPGM command. The example screens are provided in sets. The first screen in the set describes the values you can enter, the second screen presents the keywords and defaults. You can toggle between the values and keywords screens by pressing F11. The text which follows the screens describes those keywords and defaults.

In the description of the parameters all defaults are explained first and highlighted. The parameters are presented in sequence. Follow this sequence if you are entering only the parameter values without the corresponding parameter acronym.

Note: For a description of the differences between compiling RPG/400 and System/38 Environment RPG III see Appendix G, "Converting from System/38 to AS/400 System."

```
CRTRPGPGM                  Create RPG/400 Program

Type choices, press Enter.

Program . . . . . .  *CTLSPEC__    Name, *CTLSPEC
  Library . . . . . .   *CURLIB___  Name, *CURLIB
Source file . . . . .  QRPGSRC___   Name, QRPGSRC
  Library . . . . . .   *LIBL_____  Name, *LIBL, *CURLIB
Source member . . . .  *PGM_____    Name, *PGM
Generation severity level  . . .  9___  0-99
Text 'description' . . . . . .  *SRCMBRTXT_____________________________
_____

                         Additional Parameters

Source listing options . . . . .  _________  *SOURCE, *NOSOURCE, *SRC...
              + for more values   _________
Generation options . . . . . .   __________  *LIST, *NOLIST, *XREF...
              + for more values   __________
                                                                  More...
F3=Exit   F4=List   F5=Refresh   F11=Keywords   F12=Previous   F13=Prompter help
```

```
CRTRPGPGM                  Create RPG/400 Program

Type choices, press Enter.

Program . . . . . . . . . . . . PGM          *CTLSPEC__
  Library . . . . . . . . . . .                 *CURLIB___
Source file . . . . . . . . . . SRCFILE       QRPGSRC___
  Library . . . . . . . . . . .                 *LIBL_____
Source member . . . . . . . . . SRCMBR        *PGM_____
Generation severity level  . . . GENLVL       9___
Text 'description' . . . . . . . TEXT         *SRCMBRTXT_____________________
_________________

                         Additional Parameters

Source listing options . . . . . OPTION       _________
                        + for more values     _________
Generation options . . . . . . . GENOPT       __________
                        + for more values     __________
                                                                  More...
F3=Exit   F4=List   F5=Refresh   F11=Choices   F12=Previous   F13=Prompter help
```

Figure 3-2. First Set of CRTRPGPGM Prompt Screens

PGM

Specifies the library and program name by which the compiled RPG/400 program is to be known. If no library is specified, the created program is stored in the current library.

*CTLSPEC

The program name specified in positions 75 through 80 of the control specification is used.

If the program name is not specified on the control specification, but the source program is from a data base file, the member name, specified by the SRCMBR parameter, is used as the program name. If the source is not from a data base file, the program name defaults to RPGOBJ.

program-name
Enter the name by which the program will be known.

*CURLIB
The compiled program is stored in the current library. If you have not specified a current library, QGPL will be used.

library-name
Enter the name of the library where the compiled program is to be stored.

SRCFILE
Specifies the name of the source file that contains the RPG/400 source program to be compiled and the library where the source file is located.

QRPGSRC
The default source file, QRPGSRC, contains the RPG/400 source program to be compiled.

source-file-name
Enter the name of the source file that contains the RPG/400 source program to be compiled.

*LIBL
The system searches the library list to find the library where the source file is located.

*CURLIB
The current library will be used to find the source file. If you have not specified a current library, QGPL will be used.

library-name
Enter the name of the library where the source file is stored.

SRCMBR
Specifies the name of the member of the source file that contains the RPG/400 source program to be compiled. This parameter can be specified only if the source file named in the SRCFILE parameter is a data base file.

*PGM
Use the name specified by the *PGM parameter as the source file member name. The compiled program will have the same name as the source file member. If no program name is specified by the *PGM parameter, the

command uses the first member created in or added to the source file as the source member name.

`source-file-member-name`
Enter the name of the member that contains the RPG/400 source program.

GENLVL
Specifies whether or not a program object is generated depending on the severity of the errors encountered. A severity level value corresponding to the severity level of the messages produced during compilation can be specified with this parameter. If errors occur in a program with a severity level equal to or greater than the value specified in this parameter, the compilation is ended. For example, if you do not want a program generated if you have messages with a severity level of 30 or greater, specify 30 in this parameter.

9
A program object will not be generated if you have messages with a severity level greater than or equal to 9. If a severity level greater than 9 is specified, the program may contain errors that will cause unpredictable results when the program runs.

`severity-level-value:`
Enter a number, 0 through 50.

Note: The severity level value of RPG/400 messages does not exceed 50.

TEXT
Lets the user enter text that briefly describes the program and its function. The text appears whenever program information is displayed.

`*SRCMBRTXT`
The text of the source member is used.

`*BLANK`
No text appears.

`'description'`
Enter the text that briefly describes the program and it's function. The text can be a maximum of 50 characters in length and must be enclosed in apostrophes. The apostrophes are not part of the 50-character string. Apostrophes are not required if you are entering the text on the prompt screen.

OPTION
Specifies the options to use when the source program is compiled. Any or all of the options can be specified in any order. Separate the options with a blank space.

*SOURCE
Produce a source listing, consisting of the RPG/400 program input and all compile-time errors.

*NOSOURCE
A source listing is not produced. If *NOSOURCE is specified the system defaults to *NOXREF.

The acceptable abbreviation for *SOURCE is *SRC and for *NOSOURCE is *NOSRC.

*XREF
Produce a cross-reference listing and key field information table (when appropriate) for the source program.

*NOXREF
A cross-reference listing is not produced.

*NOXREF is the default when either *NOSOURCE or *NOSRC is specified.

*GEN
Create a program object that can be run after the program is compiled.

*NOGEN
Do not create a program object.

*NODUMP
Do not dump major date areas when an error occurs during compilation.

*DUMP
Dump major data areas when an error occurs during compilation.

*NOSECLVL
Do not print second-level message text on the line following the first-level message text.

*SECLVL
Print second-level message text.

GENOPT
Specifies the options to use to create the program object: the printing of the IRP (intermediate representation of a program), a cross-reference listing of objects defined in the IRP, an attribute listing from the IRP, and the program template. GENOPT reserves a program patch area, and specifies optimization of a program for more efficient running. These results may be useful if a problem occurs when trying to run the compiled program. Any or all of the options can be specified in any order. Separate the values with a blank. For

a description of the GENOPT parameter and the information it provides, see "Compiler Debugging Options" on page A-5 in Appendix A, "RPG Compiler and Auto Report Program Service Information."

*NOOPTIMIZE
Do not process program optimization.

*OPTIMIZE
Process program optimization. With *OPTIMIZE the compiler generates a program for more efficient processing and one that will possibly require less storage. However, specifying *OPTIMIZE can substantially increase the time required to create a program. Existing program objects may be optimized with the CL command CHGPGM.

```
CRTRPGPGM                  Create RPG/400 Program

Type choices, press Enter.

Print file . . . . . .  QSYSPRT___    Name
  Library . . . . . .    *LIBL_____   Name, *LIBL, *CURLIB
Replace program . . . .  *YES         *YES, *NO
User profile . . . . . .  *USER_       *USER, *OWNER
Authority . . . . . .    *CHANGE___   Name, *CHANGE, *ALL, *USE...
Phase trace . . . . . .  *NO_         *NO, *YES
Intermediate text dump . . . . .  *NONE_____________________________
_____
Snap dump . . . . . .    *NONE_____________________________________
_____
Codelist . . . . . .     *NONE_____________________________________
_____
Ignore decimal data error . . .  *NO_         *NO, *YES


                                                              Bottom
F3=Exit   F4=List   F5=Refresh   F11=Keywords   F12=Previous   F13=Prompter help
```

Figure 3-3 (Part 1 of 2). Second Set of CRTRPGPGM Prompt Screens

```
CRTRPGPGM                  Create RPG/400 Program

Type choices, press Enter.

Print file . . . . . . . . . . . PRTFILE       QSYSPRT___
  Library . . . . . . . . . . .                  *LIBL____
Replace program . . . . . . . . . REPLACE      *YES
User profile . . . . . . . . . . USRPRF        *USER_
Authority . . . . . . . . . . . . AUT          *CHANGE___
Phase trace . . . . . . . . . . . PHSTRC       *NO_
Intermediate text dump . . . . . ITDUMP        *NONE_____________________
_____________________
Snap dump . . . . . . . . . . . . SNPDUMP      *NONE_____________________
_____________________
Codelist . . . . . . . . . . . . CODELIST      *NONE_____________________
_____________________
Ignore decimal data error . . . IGNDECERR      *NO_


                                                                    Bottom
F3=Exit   F4=List   F5=Refresh   F11=Choices   F12=Previous   F13=Prompter help
```

Figure 3-3 (Part 2 of 2). Second Set of CRTRPGPGM Prompt Screens

PRTFILE

Specifies the name of the file where the compiler listing is to be placed and the library where the file is located. If you specify a file whose record length is less than 132, information will be lost.

QSYSPRT

If a file name is not specified, the compiler listing is placed in the IBM-supplied file, QSYSPRT. The file QSYSPRT has a record length of 132.

file-name

Enter the name of the file where the compiler listing is to be placed.

*LIBL

The system searches the library list to find the library in which the file is located.

*CURLIB

The current library will be used to find the file. If you have not specified a current library, QGPL will be used.

library-name

Enter the name of the library where the file is located.

REPLACE

Specifies if a new program object will be created without first deleting any existing object.

*YES A new program object will be created and any existing object of the same name in the specified library will be moved to library QRPLOBJ.

*NO A new program object will not be created if an object of the same name already exists in the specified library.

USRPRF

Specifies the user profile the compiled RPG/400 program runs under. This profile controls which objects can be used by the program (including what authority the program has for each object).

*USER
The program runs under the user profile of the program's user.

*OWNER
The program runs under the user profiles of both the program's owner and user. The collective sets of object authority in both user profiles are used to find and access objects while the program is running. Any objects that are created during the program are owned by the program's user.

Note: The USRPRF parameter reflects the security requirements of your installation. The security facilities available on the AS/400 System are described in detail in the *Security Concepts and Planning* and the *Programming: Control Language Reference.*

AUT

Specifies what authority for the program and its description is being granted to the public. The authority can be altered for all or for specified users after program creation with the CL commands GRTOBJAUT (Grant Object Authority) or RVKOBJAUT (Revoke Object Authority). (For further information on these commands, see the *Programming: Control Language Reference.*)

*CHANGE
The public has only operational rights for the compiled program. Any user can run the program, but cannot change it.

*USE
The public can run the compiled program, but cannot debug or change it.

*ALL
The public has complete authority for the program.

*EXCLUDE
The public cannot use the program.

authorization-list name
Name of an authorization list to which the program is added. For a description of the authorization list and how to create it see the *Programming: Control Language Reference.*

Note: Use the AUT parameter to reflect the security requirements of your installation. The security facilities available are described in detail in the *Security Concepts and Planning*.

PHSTRC

Specifies whether information about compiler phases is provided on the listing. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

*NO
: Do not provide information about compiler phases.

*YES
: Provide information about compiler phases.

ITDUMP

This parameter specifies whether a dynamic listing of intermediate text for one or more specified phases is to be printed at compile time as each IT record is being built. This parameter also specifies whether a flow of the major routines run in one or more specified phases is to be printed. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

*NONE
: No intermediate dump is produced.

phase-name
: Enter the last two characters of phase name. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

SNPDUMP

Specifies whether the major data areas are to be printed after the running of one or more specified phases. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

*NONE
: No snap dump is produced.

phase-name
: Enter the last two characters of phase name. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

CODELIST

Specifies whether a dynamic listing of the IRP is to be printed during compilation of one or more specified phases of the source program. See

Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

*NONE
Do not produce a code listing for each of the code generating phases run.

*ALL
Produce a code listing for each of the code generating phases run.

phase-name
Enter the last 2 characters of phase name. See Appendix A, "RPG Compiler and Auto Report Program Service Information" for a detailed explanation of this parameter.

IGNDECERR
Specifies whether decimal data errors detected by the system are ignored by the program.

*NO
Do not ignore decimal data errors. The result of the operation being processed when the error occurs is unknown. The compiler generates an error message on the compiler listing to notify the user that this option was specified. Incorrect results that occur while a program is running when this option is specified are the user's responsibility.

*YES
Ignore decimal data errors.

Compiling under the System/38 Environment

You can also compile an RPG/400 source program from the System/38 Environment. Here, you would call the compiler with the same commands as you would use in the AS/400 Native Environment (CRTRPGPGM to call up the RPG/400 compiler, and CRTRPTPGM to call up the auto report). To compile a program from the System/38 Environment, use the CL command CALL QCL to call up the System/38 Environment before entering the CRTRPGPGM command.

For more information on the differences between RPG/400 in AS/400 Native Environment and in the System/38 Environment see Appendix G, "Converting from System/38 to AS/400 System."

For further information about programming in the System/38 Environment, refer to the *System/38 RPG III Reference Manual and Programmer's Guide.*

Chapter 4. Error Messages, Testing and Debugging

This chapter describes error messages you may receive from RPG, how to interpret them, and how they can be displayed and printed. It also describes testing and debugging an RPG/400 program using functions provided by RPG/400 and OS/400.

OS/400	RPG
• Test library • Breakpoints • Traces	• DEBUG operation code • DUMP operation code

The OS/400 functions allow you to use CL commands to test programs while protecting your production files, and let you observe and debug operations as a program runs. No special source code is required to use the OS/400 functions. The RPG/400 functions can be used independently of the OS/400 functions or in combination with them to:

- Debug a program, or
- Produce a formatted dump of indicator settings and the contents of fields, data structures, arrays, and tables.

Special source code is required to use the RPG/400 DEBUG and DUMP operation codes. A formatted dump can also be obtained by a user's response to a run-time message.

A file information data structure and a program status data structure can provide additional debugging information. These data structures are described later in this chapter. Following this is information on exception/error handling.

Using, Displaying, and Printing Messages

Using Messages

This manual refers to messages you receive during both compile and run-time. These messages are displayed on your screen or printed on your compiler listing. There are no message manuals for this product.

Each message contains a minimum of three parts as illustrated below:

```
A 10
B Syntax of Program-Identification entry is invalid.  Defaults to RPGOBJ.
C Cause:The Program-Identification entry (positions 75-80) of a control
  specification has an invalid syntax: the first character is not
  alphabetic or it is not left-justified, or it contains embedded blanks
  or special characters.  Defaults to RPGOBJ.

  Recovery:Specify RPGOBJ or a valid entry (positions 75-80) for the
  Program-Identification option.  Recompile.
```

A A number indicating the severity of the error. Numbers 40 and above are reserved for run-time errors.

Severity	Meaning
00	An informational message displayed during entering, compiling, and running. This level is used to convey information to the user. No error has been detected and no corrective action is necessary.
10	A warning message displayed during entering, compiling and running. This level indicates that an error was detected but is not serious enough to interfere with the running of the program. The results of the operation are assumed to be successful.
20	An error message displayed during compiling. This level indicates that an error was made, but the compiler is taking a recovery that might yield the desired code. The program may not work as the author intended.
30	A severe error message displayed during compiling. This level indicates that an error too severe for automatic recovery was detected. Compilation is completed, but running the program cannot be attempted.
40	An abnormal end of program or function message displayed during running. This level indicates an error that forces cancellation of processing. The operation may have ended because it was unable to handle valid data, or possibly because the user cancelled it.
50	An abnormal end of job message displayed during running. This level indicates an error that forces cancellation of job. The job may have ended because a function failed to perform as required, or possibly because the user cancelled it.
99	A user action to be taken during running. This level indicates that some manual action is required of the operator, such as entering a reply, changing diskettes, or changing printer forms.

B The text you see online or on a listing. This text is a brief, generally one sentence, description of the problem.

C The text you see online when you press F4 from the screen with the first-level text. This text will be printed on your listing if you specify *SECLVL in your compile-time options. The IBM-supplied default for this option is *NOSECLVL. This text contains an expanded description of the message and a section detailing the correct user response.

You can enter D to obtain RPG formatted dump, S to obtain system dump, C to cancel, or G to continue processing at *GETIN.

Displaying and Printing Messages

To display or print a particular message or messages, use the DSPMSGF or DSPMSGD commands. These commands are described in the *Programming: Control Language Reference*. The library name for your product is QRPG, and the file names are QRPGMSG and QRPGMSGE. The run-time messages are stored in a file called QRPGMSGE in library QSYS. The compile time messages are in file QRPGMSG in library QRPG.

Note: If you have any comments or suggestions concerning the messages, please use the Reader Comment Form included with this manual and send them to us.

How to Run an RPG/400 Program

There are many ways to run an RPG/400 program depending on how the program is written and who is using the program. See the *CL Programmer's Guide* for the various ways to run an RPG/400 program. The three most common ways of running an RPG/400 program are through:

- A high-level language CALL statement or operation
- An application-oriented menu
- A user-created command.

The CL statement CALL can be part of a batch job, it can be entered interactively by a work station user, or it can be included in a CL program. An example is CALL PAYROLL. PAYROLL is the name of either a CL program or an RPG/400 program that is called and then run. An RPG/400 program can call another program with the CALL operation code (see Chapter 11, "Communicating with Objects in the System").

Another way to run an RPG/400 program is through an application-oriented menu. The work station user can request an application-oriented menu and then select an option that will call the appropriate program. Figure 4-1 is an example of an application-oriented menu:

```
                    PAYROLL DEPARTMENT MENU

1.   Inquire into employee master

2.   Change employee master

3.   Add new employee

4.   Return


Option:_
```

Figure 4-1. Example of an Application-Oriented Menu

This menu is normally written as a CL program in which each option calls a separate RPG/400 program. When an RPG/400 program is ended, the system returns

control to the calling program or to the user. This could be a work station user, a CL program (such as the menu handling program), or another RPG/400 program.

You can also create a command yourself to run an RPG/400 program by using a command definition. (See the *Programming: Control Language Programmer's Guide* for a description of how to define a command.) For example, you can create a command, PAY, which calls a PAYROLL program. A user-created command can be entered into a batch job, or it can be entered interactively by a work station user.

The OS/400 functions allow you to use CL commands to test programs while protecting your production files, and let you observe and debug operations as a program runs. No special source code is required to use the OS/400 functions. The RPG/400 functions can be used independently of the OS/400 functions or in combination with them to:

- Debug a program, or
- Produce a formatted dump of indicator settings and the contents of fields, data structures, arrays, and tables.

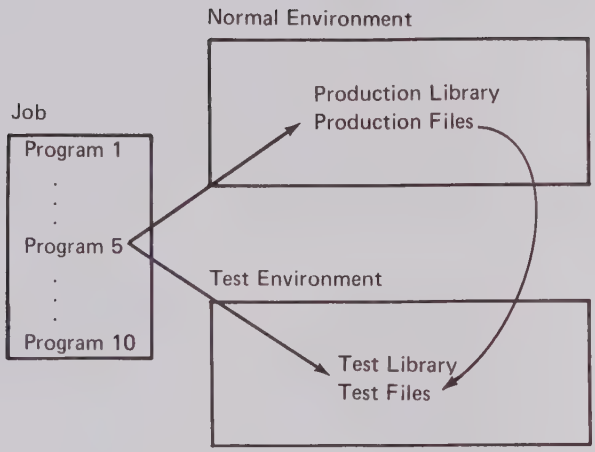
Special source code is required to use the RPG/400 DEBUG and DUMP operation codes. A formatted dump can also be obtained by a user's response to a run-time message.

A file information data structure and a program status data structure can provide additional debugging information. These data structures are described later in this chapter.

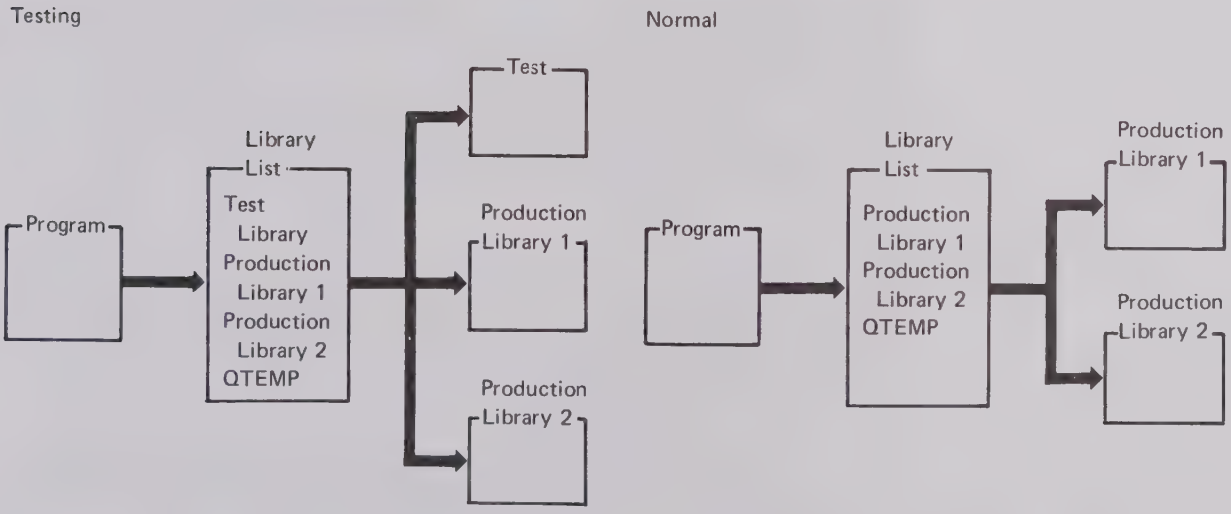
Using a Test Library

The basic concept of testing and debugging is that of a separate testing environment. Programs running in a normal operating environment can read, update, and write records that are in either test or production libraries. Programs running in a test environment can read, update, and write records in either test or production libraries. However, to prevent data base files in production libraries from being modified unintentionally, you can specify the UPDPROD(*NO) option on the CL command STRDBG (Start Debug) (see the *CL Reference*).

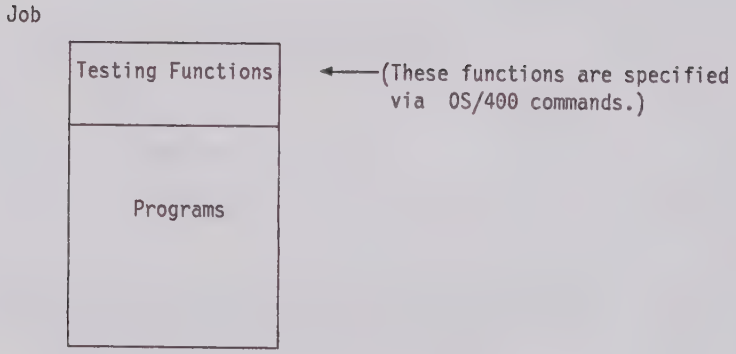
On the AS/400 system, you can copy production files into the test library or you can create special files for testing in this library. A test copy of a file and its production copy can have the same name if the files are in different libraries. You can use the same file name in the program for either testing or normal processing.



For testing, you must place the test library name ahead of the production library name in the library list for the job that contains the program to be tested. For normal running, the production library should be the only library named in the library list for that job (that is, the test library should not be named).



No special statements for testing are contained within the program being tested. The same program being tested can be run normally without modifications. All testing functions are specified within the job that contains the program and not within the program.



Testing functions apply only to the job in which they are specified. A program can be used concurrently in two jobs: one job that is in a test environment and another job that is in a normal processing environment.

OS/400 testing functions let you interact with a program as it runs to observe its processing. These functions include using breakpoints and traces.

Using Breakpoints

Breakpoints can be used to stop your program at a specified breakpoint. This breakpoint can be a statement number or a label in your program. If you use a label as a breakpoint rather than a statement number, the label can be:

- On a TAG statement in the program
- Associated with a step in the RPG/400 program cycle (for example, *TOTC indicates the beginning of total calculations and *TOTL indicates the beginning of total output)
- Associated with a function done by your RPG/400 program (for example, SQRT indicates the square root function).

When a breakpoint is encountered for an interactive job, the system displays the breakpoint at which the program has stopped and, if requested, the values of program variables. After you get this information (in a display) you can go to a Command Entry Screen and then enter CL commands to request other functions (such as displaying or changing a variable, adding a breakpoint, or adding a trace).

When a breakpoint is encountered for a batch job, a breakpoint program can be called. You must create this breakpoint program to handle the breakpoint information.

Example of Using Breakpoints

Figure 4-2 shows a source listing of a sample RPG/400 program, DBGPGM, and the CL commands that add breakpoints at statements 1200 and 1500. The value of variable *IN is displayed when the breakpoint at statement 1200 is reached, and the value of variables FLD1 and PART are displayed when the breakpoint at statement 1500 is reached.

CL Commands

```
STRDBG  PGM(EXAMPLES/DBGPGM)
ADDBKP  STMT(1200) PGMVAR((*IN))
ADDBKP  STMT(1500) PGMVAR((FLD1) (PART))
```

All CL commands are explained in the *Programming: Control Language Reference.*

```
5728RG1 R01 M00 880715          IBM AS/400 RPG/400              QGPL/DBGPGM       01/15/88  13:42:19     Page       2
SEQUENCE                                                                    IND   DO   LAST       PAGE   PROGRAM
NUMBER    *...1....+....2....+....3....+....4....+....5....+....6....+....7...*  USE   NUM  UPDATE     LINE   ID
                        S o u r c e   L i s t i n g
          H                                                                                        *****
       100 FTESTX   IF  F      5           DISK                                                  01/01/88
       200 FTESTA   UF  F     10           DISK                                                  01/01/88
       300 ITESTX   NS  01                                                                       01/01/88
       400 I                                      1   5 PART                                     01/01/88
       500 ITESTA   NS  02                                                                       01/01/88
       600 I                                      1   5 FLD1                                     01/01/88
       700  ******************************************************                               01/01/88
       800  *       MAINLINE                                                                     01/01/88
       900  ******************************************************                               01/01/88
      1000 C           LOOP      TAG                                                           01/01/88
      1100 C                     READ TESTX                    66                    3           01/01/88
      1200 C   66                GOTO ENDPGM                                                   01/01/88
      1300 C                     READ TESTA                    67                    3           01/01/88
      1400 C  N67                MOVE PART      FLD1                                           01/01/88
      1500 C  N67                EXCPTMAST                                                     01/01/88
      1600 C  N66                GOTO LOOP                                                     01/01/88
      1700 C           ENDPGM    TAG                                                           01/01/88
      1800 C                     SETON                     LR                  1               01/01/88
      1900 OTESTA  E                MAST                                                       01/01/88
      2000 O                        FLD1       5                                              01/01/88
             * * * * *  E N D   O F   S O U R C E  * * * * *
```

Figure 4-2. Sample RPG/400 Program DBGPGM

The first breakpoint is used to tell you where you are in the program. Figure 4-3 shows the display as a result of reaching the first breakpoint.

```
                          Display Breakpoint

 Statement/Instruction . . . . . :  1200 /004A
 Program . . . . . :  DBGPGM
 Recursion level . . . . . :  1
 Start position  . . . . . :  1
 Format  . . . . . :  *CHAR
 Length  . . . . . :  *DCL


 Variable  . . . . . :  *IN
   Lower/upper bounds  . . . . . :  (1:99)
   Type  . . . . . :  CHARACTER
   Length  . . . . . :  1
   Element     -------------------- Values ------------------- .
          1  '1'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         11  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         21  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         31  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'              +

Press Enter to continue.

F3=Exit program   F10=Command entry
```

```
                          Display Breakpoint

 Statement/Instruction . . . . . :  1200 /004A
 Program . . . . . :  DBGPGM
 Recursion level . . . . . :  1
 Start position  . . . . . :  1
 Format  . . . . . :  *CHAR
 Length  . . . . . :  *DCL


         41  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         51  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         61  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         71  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         81  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'
         91  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'  '0'

Press Enter to continue.

F3=Exit program   F10=Command entry
```

Figure 4-3. First Breakpoint Display for DBGPGM

Figure 4-4 shows the display as a result of reaching the second breakpoint.

```
                        Display Breakpoint

 Statement/Instruction . . . . . :   1500 /0060
 Program . . . . . :   DBGPGM
 Recursion level . . . . . :   1
 Start position  . . . . . :   1
 Format  . . . . . :   *HEX
 Length  . . . . . :   *DCL


 Variable  . . . . . :   FLD1
   Type  . . . . . :   CHARACTER
   Length  . . . . . :   5
   * . . . +. . . . 1 . . . . + . .  *...+....1....+.
   404000063F                        '    '

 Variable  . . . . . :   PART
   Type  . . . . . :   CHARACTER
   Length  . . . . . :   5                                +

Press Enter to continue.

F3=Exit program   F10=Command entry
```

```
                        Display Breakpoint

 Statement/Instruction . . . . . :   1500 /0060
 Program . . . . . :   DBGPGM
 Recursion level . . . . . :   1
 Start position  . . . . . :   1
 Format  . . . . . :   *HEX
 Length  . . . . . :   *DCL

   * . . . +. . . . 1 . . . . + . .  *...+....1....+.
   404000063F                        '    '









Press Enter to continue.

F3=Exit program   F10=Command entry
```

Figure 4-4. Second Breakpoint Display for DBGPGM

At this point you could change the value of one of these variables to alter how your program runs. After getting the command entry screen by pressing F10, you can use the CL command CHGPGMVAR (Change Program Variable) to change the value of a variable.

This command is explained in the *Programming: Control Language Reference.*

Considerations for Using Breakpoints

You should know about the following characteristics of breakpoints before using them:

- If a breakpoint is part of a conditional statement, that breakpoint request is processed even if the conditions are not met.
- If a breakpoint is bypassed by a GOTO operation, that breakpoint request is not processed.
- Some statements that are not processed do not represent a definite position in the logic flow of your program. Avoid putting breakpoints on PLIST, PARM, KLIST, KFLD and DEFN operations.
- When a breakpoint is requested for a statement, the breakpoint occurs before that statement is run.
- When a breakpoint is requested for a statement that is not processed such as a TAG operation, the breakpoint is set on the next statement.
- Breakpoint functions are specified using CL commands. You can use CL commands to add breakpoints to programs, display breakpoint information, remove breakpoints from programs and start a program after a breakpoint has been displayed. Refer to the *Programming: Control Language Reference* for descriptions of these commands and for a further description of breakpoints.
- Input fields not used in your program cannot be specified in the PGMVAR parameter of the debug commands. However, the entire input or output buffer for a record may be displayed by using the variable name ZZnnBIN (input buffer) or ZZnnBOUT (output buffer). The nn value is the sequence number corresponding to the order in which the files are defined in your specifications. This number also appears in the cross reference section of the compiler listing. Thus the input buffer for the second file in your program may be displayed by specifying PGMVAR (ZZ02BIN).

Using a Trace

You can use a trace to record the statements that were run in a program and the values of the variables used in the statements.

To use a trace you specify what statements and variables the system should trace. You can also specify that variables be traced only when their values change.

You can specify a trace of one statement in a program, a group of statements in a program, or an entire program.

You must request a display of the traced information. The display shows the sequence in which the statements were run and, if requested, the values of variables used in the statements.

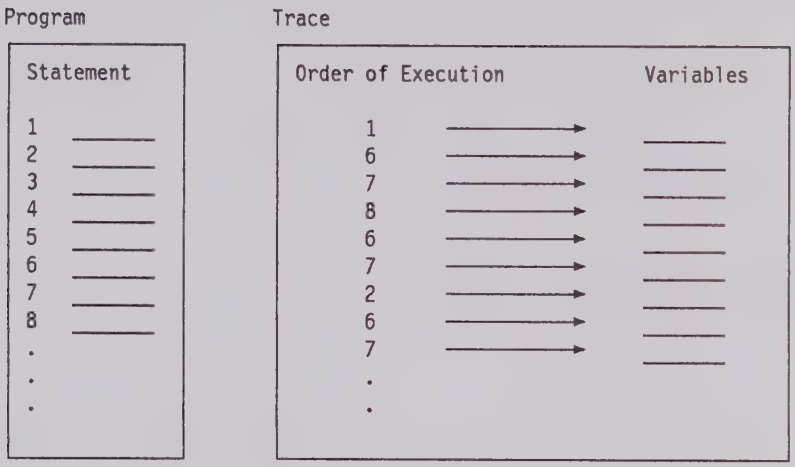


Figure 4-5. Program Statements and Order of Execution

Example of Using a Trace

Figure 4-2 on page 4-7 shows a portion of a listing of RPG/400 program, DBGPGM. The CL command that adds a trace of statements 1000 through 1800 in that program is:

```
ADDTRC STMT(1000 1800)
```

All CL commands are explained in the *Programming: Control Language Reference.*

Figure 4-6 is an example of a display of the traced information. The CL command to display this information is:

```
DSPTRCDTA OUTPUT(*)
```

The CL command DSPTRCDTA (Display Trace Data) is explained in the *Programming: Control Language Reference.*

```
                        Display Trace Data

                  Statement/
Program           Instruction        Recursion Level      Sequence Number
DBGPGM            1000                     1                     1
DBGPGM            1200                     1                     2
DBGPGM            1300                     1                     3
DBGPGM            1400                     1                     4
DBGPGM            1500                     1                     5
DBGPGM            1600                     1                     6
DBGPGM            1000                     1                     7
DBGPGM            1200                     1                     8
DBGPGM            1800                     1                     9




Press Enter to continue.
F3=Exit    F12=Previous
```

Figure 4-6. Trace Data Display for DBGPGM

Considerations for Using a Trace

You should know about the following characteristics of traces before using them:

- A conditioned statement is recorded in the trace even if the conditions are not met.
- Statements bypassed by GOTO operations are not included in the trace.
- Trace functions are specified with CL commands in the job that contains the traced program. These functions include adding trace requests to a program, removing trace requests from a program, removing data collected from previous traces, displaying trace information, and displaying the traces that have been specified for a program.

Using the DEBUG Operation Codes

You can code one or more DEBUG operation codes among your RPG/400 calculations to help you debug a program that is not working properly. Whenever the DEBUG operation is processed, one or two records with debugging information are provided: the first record contains a list of all indicators that are set on at the time the DEBUG operation was encountered; the second record is optional and shows the contents of the result files specified for the DEBUG operation.

The DEBUG operation can be coded at any point or at several points in the calculation specifications. The output records are written whenever the DEBUG operation occurs.

You should know about the following characteristics of the DEBUG operation code before using it:

- The DEBUG operation runs (is active) only if position 15 of the control specification contains 1.
- If the DEBUG operation is conditioned, it occurs only if the conditions are met.
- If a DEBUG operation is bypassed by a GOTO operation, the DEBUG operation does not occur.

You can apply the OS/400 testing and debugging functions to programs that use DEBUG operations; a breakpoint can be on a DEBUG operation, and a DEBUG operation can be traced.

Using the RPG/400 Formatted Dump

To obtain an RPG/400 formatted dump (printout of storage) for a program while it is running, you can code one or more DUMP operation codes in calculations, or you can respond to a run-time message with a D or F option. It is also possible to automatically reply to make a dump available. Refer to the "System Reply List" discussion in the *Programming: Control Language Programmer's Guide.*

The formatted dump includes field contents, data structure contents, array and table contents, the file information data structure, and the program status data structure. The dump is written to the file called QPPGMDMP. (A system abnormal dump is written to the file QPSRVDMP.)

If the user responds to an RPG/400 run-time message with an F option, the dump also includes the hexadecimal representation of the open data path (ODP, a data management control block). If position 15 of the control specification contains a 1, the F option also provides a list of compiler-generated fields.

Information from the file information data structure (INFDS) is provided for each file in the program. However, not all the information that is contained in the INFDS is printed in the dump. Remember that to use any information from the INFDS in your program, you must define the INFDS in your program.

You should know about the same characteristics for using the DUMP operation as those previously described for the DEBUG operation.

Figure 4-7 on page 4-14 shows an example of an RPG/400 formatted dump.

Note

Only selected pages of an RPG/400 formatted dump are presented below.

```
RPG/400 FORMATTED DUMP
Program Status Area:
Program Name . . . . . :  QGPL/SAMPLE      A
Program Status . . . . . :  00000          B
Previous Status . . . . . :  00000         C
Statement in Error . . . . . :  00000000   D
RPG Routine . . . . . :  *DETC             E
Number of Parameters . . . . . :  000
Message Type . . . . . :                   F
MI Statement Number . . . . . :            G
Additional Message Info . . . . . :
Message Data . . . . . :
Last File Used . . . . . :  QSYSPRT
Last File Status . . . . . :  01235
        Error in PRTCTL entries occurred in  (C G S D).   H
Last File Operation . . . . . :  OPEN I
Last File Routine . . . . . :  *INIT
Last File Statement . . . . . :  *INIT
Last File Record Name . . . . . :
Job Name . . . . . :  E53
User Name . . . . . :  QPGMR
Job Number . . . . . :  000811
Date Entered System . . . . . :  011588
Date Started . . . . . :  011588
Time Started . . . . . :  111143           I
Compile Date . . . . . :  011588
Compile Time . . . . . :  111125
Compiler Level . . . . . :  0001
Source File . . . . . :  QRPGSRC
  Library . . . . . :  QGPL
Member . . . . . :  SAMPLE
```

Program Status Information

Figure 4-7 (Part 1 of 8). RPG Formatted Dump

```
RPG/400 FORMATTED DUMP

File . . . . . :  FILEIN1
File Open  . . . . . :  YES
File at EOF  . . . . . :  YES
Commit Active  . . . . . :  NO
File Status  . . . . . :  00011
            RPG0011 End of file (input).
File Operation . . . . . :  READ R
File Routine . . . . . :  *DETC
Statement Number . . . . . :  2500
Record Name  . . . . . :  FILEA
Message Identifier . . . . . :
MI Instruction Number  . . . . . :
ODP type . . . . . :  DB
File Name  . . . . . :  FILEIN1
   Library . . . . . :  QGPL
Member . . . . . :  FILEIN1
Record Format  . . . . . :
Primary Record Length  . . . . . :  45
Secondary Record Length  . . . . . :  0
Input Block Length . . . . . :  4125
Output Block Length  . . . . . :  0
Device Class . . . . . :  '0000'X
Lines per Page . . . . . :  0
Columns per Line . . . . . :  0
Number of Records in File  . . . . . :  0
Access Type  . . . . . :  ARRIVAL SEQ
Allow Duplicate Keys . . . . . :  NO
Source File  . . . . . :  NO
UFCB Parameters  . . . . . :  'A2000000000000500000'X
UFCB Overrides . . . . . :  '0000000000000000000'X
Records to Transfer  . . . . . :  74
Number of Puts . . . . . :  0
Number of Gets . . . . . :  0
Number of Put/Gets . . . . . :  0
Number of other I/O  . . . . . :  0
Current Operation  . . . . . :  '4040'X
Device Class . . . . . :  '4040'X
Device Name  . . . . . :
Length of Last Record  . . . . . :  0
DDS Information  . . . . . :
Relative Record Number . . . . . :  0
Records Transferred  . . . . . :  0
Current Line Number  . . . . . :  0
Input Buffer:
    0000   80000000  00000000  0007C00D  BD000880  0000004A  0037002D  40404040  40404040       *                *
    0020   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    0040   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    0060   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    0080   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    00A0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    00C0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    00E0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    0100   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    0120   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    0140   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
    0160   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040       *                *
```

Figure 4-7 (Part 2 of 8). RPG Formatted Dump

```
RPG/400 FORMATTED DUMP
    0180   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    01A0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    01C0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    01E0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0200   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0220   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0240   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0260   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0280   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    02A0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    02C0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    02E0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0300   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0320   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0340   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0360   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0380   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    03A0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    03C0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    03E0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0400   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0420   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0440   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0460   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0480   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    04A0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    04C0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    04E0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0500   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0520   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0540   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0560   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0580   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    05A0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    05C0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    05E0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0600   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0620   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0640   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0660   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0680   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    06A0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    06C0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    06E0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0700   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0720   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0740   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0760   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0780   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    07A0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    07C0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    07E0   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0800   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0820   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0840   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
    0860   40404040  40404040  40404040  40404040  40404040  40404040  40404040  40404040      *
```

Figure 4-7 (Part 3 of 8). RPG Formatted Dump

```
RPG/400 FORMATTED DUMP
File . . . . . :  QSYSPRT
File Open  . . . . . :  YES
File at EOF  . . . . . :  NO
Commit Active  . . . . . :  NO
File Status  . . . . . :  01235
          Error in PRTCTL entries occurred in  (C G S D).   J
File Operation . . . . . :  OPEN I
File Routine . . . . . :  *INIT
Statement Number . . . . . :  *INIT
Record Name  . . . . . :
Message Identifier . . . . . :
MI Instruction Number  . . . . . :
ODP type . . . . . :  SP
File Name  . . . . . :  QSYSPRT
   Library . . . . . :  QSYS
Member . . . . . :  Q713784701
Record Format  . . . . . :
Spool File . . . . . :  Q04079N001
   Library . . . . . :  QSPL
Spool File Number  . . . . . :  19
Primary Record Length  . . . . . :  132
Secondary Record Length  . . . . . :  0
Input Block Length . . . . . :  0
Output Block Length  . . . . . :  132
Device Class . . . . . :  PRINTER
Lines per Page . . . . . :  10
Columns per Line . . . . . :  132
Number of Records in File  . . . . . :  0
Access Type  . . . . . :  0
Allow Duplicate Keys . . . . . :  NO
Source File  . . . . . :  NO
UFCB Parameters  . . . . . :  'A4121000000000000000'X
UFCB Overrides . . . . . :  '00000000000000000000'X
Number of Puts . . . . . :  0
Number of Gets . . . . . :  0          K
Number of Put/Gets . . . . . :  0
Number of other I/O  . . . . . :  0
Current Operation  . . . . . :  '4040'X
Device Class . . . . . :  '4040'X
Device Name  . . . . . :
Length of Last Record  . . . . . :  0
DDS Information  . . . . . :
Relative Record Number . . . . . :  0
Current Line Number  . . . . . :  0
Input Buffer:
    0000   E2C1D4D7  D3C54040  4040D8C7  D7D34040  40404040  00000000  00000000  00000000        *SAMPLE    QGPL                *
    0020   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000        *                              *
    0040   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000        *                              *
    0060   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000        *                              *
    0080   00000000                                                                               *                              *
Output Buffer:
    0000   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000        *                              *
    0020   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000        *                              *
    0040   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000        *                              *
    0060   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000        *                              *
    0080   00000000                                                                               *                              *
```

File information.
This information is
repeated for each file
in the program. For a
detailed description of these
entries see
"File Information Data Structure" on page 4-23.

Figure 4-7 (Part 4 of 8). RPG Formatted Dump

```
RPG/400 FORMATTED DUMP
Open Data Path:          L
    0000   64800000  000010A4  00001100  000000B0  00000140  000001C6  00000280  000002C0      *                    F       *
    0020   00000530  00000000  00000000  00000140  00000000  00000000  00000000  00000000      *                            *
    0040   00000000  00000016  0007C00D  BA0019FF  00000000  00000000  00000000  00000000      *                            *
    0060   80000000  00000A80  0007C001  DB001710  00000000  00000000  00000000  00000000      *                            *
    0080   80000000  00000000  0007C00D  BA001120  01900000  00010000  00000084  00000000      *                            *
    00A0   08000000  00000000  00000000  00100000  E2D7D8E2  E8E2D7D9  E3404040  D8E2E8E2      *                SPQSYSPRT   QSYS*
    00C0   40404040  4040D8F0  F4F0F7F9  D5F0F0F1  D8E2D7D3  40404040  40400013  00840000      *     Q04079N001QSPL         *
    00E0   D8F7F1F3  F7F8F4F7  F0F10000  00000000  00840002  00000000  0A008400  00000000      *Q713784701                  *
    0100   0000D5A4  12100000  00000000  00000000  00000000  00000000  00000100  09000000      *  N                         *
    0120   0005A000  5CD54040  40404040  40400001  00000000  00000000  00000000  00000000      *     N                      *
    0140   00010001  5CD54040  40404040  40400000  06700000  00000000  00450045  00450045      *     N                      *
    0160   07A10045  00450045  00700045  00450045  00450045  00450045  002F0030  00040005      *                            *
    0180   5CD54040  40404040  40400208  00000000  20000000  00000000  00000000  00000000      * N                          *
    01A0   00000000  00000001  C2200000  00059A00  00000000  00000000  00000000  00000000      *        B                   *
    01C0   00000000  00000090  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    01E0   00000000  02080000  00000000  00000000  00000084  00000000  00000000  00000000      *                            *
    0200   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    0220   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    0240   00000000  00000000  00000000  00000000  00000000  00000001  00000001  00000000      *                            *
    0260   00000000  00000000  00000000  00000000  00000000  00000000  F0F0F0F0  00000000      *                    0000    *
    0280   00000001  00300000  00000000  00003000  00000000  0000001C  000502B5  D90019FF      *                       R    *
    02A0   00000000  00000000  00000000  00000000  41100000  00000000  00000000  00000000      *                            *
    02C0   80000000  0000004F  0007C00D  BB001824  80000000  00000000  0007C00D  BB000860      *                            *
    02E0   80000000  00000000  0007C00D  BB000860  80000000  00000000  0007C00D  BB001838      *                            *
    0300   00000000  00000700  000502DB  2700195F  00000000  00000000  0FEF0000  30700000      *                            *
    0320   0FCC00D6  005E0000  0001DEC0  00000000  00000000  00000000  00000000  00010000      *  0                         *
    0340   000186A0  C6000000  5CD1D6C2  40404040  40404040  40404040  40404040  5CE2E3C4      *    F   JOB                 STD*
    0360   40404040  40400000  00000000  000001F0  80000000  000081F0  0007C00D  BB0003B0      *              0     0        *
    0380   80000000  00000000  0007C00D  BB000860  80000000  00000000  0007C00D  BB000860      *                            *
    03A0   80000000  00000000  0007C00D  BB000460  80000000  00000000  0007C00D  BB000576      *                            *
    03C0   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    03E0   00000000  00000000  00000000  00000000  D8F0F4F0  F7F9D5F0  F0F10048  D8E2D7D3      *              Q04079N001  QSPL*
    0400   40404040  40400049  00000000  00000000  00004040  40404040  40404040  00010100      *                            *
    0420   F0F1F0F0  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *0100                        *
    0440   00010FEF  7FFF0001  00840004  00010001  D8E2E8E2  D7D9E340  4040FFF5  00000000      *              QSYSPRT    5  *
    0460   00000000  00000000  00000000  00000000  FFF30000  00000000  00000000  00000000      *               3            *
    0480   00000000  0000000C  C0000A00  84000E00  06000F00  09001000  00110000  12C04000      *                            *
    04A0   13007FFF  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    04C0   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    04E0   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    0500   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    0520   00000000  00000000  00000000  00000000  006A0800  00000000  00000650  00000000      *                            *
    0540   00000000  0000008C  000002C0  00005C00  0002FFFF  00010001  00015CC3  C8C1D5C7      *                     CHANG*
    0560   C5404040  00000000  00010000  00000000  00000000  00000000  00000000  00000000      *E                           *
    0580   00000080  00000000  00000000  00000000  D8E2E8E2  D7D9E340  404000AE  0601000A      *              QSYSPRT       *
    05A0   00840009  0D405CC4  C5E5C440  40404040  40404040  40404040  40405CD7  D9E3C9D4      *      DEVD                  PRTIM*
    05C0   C7404040  40404040  40404040  4040060A  060F100F  00000000  00000000  00000000      *G                           *
    05E0   00000000  009E0000  01001000  00C1D7D7  D3D7C7D4  40000000  00000000  00000000      *           APPLPGM          *
    0600   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    0620   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
    0640   D7C94040  40404040  00000000  00000000  A4121000  00000000  00000000  40000000      *                           C*
    0660   00000000  00000000  00000000  00000000  80000000  00000018  0007C00D  BB000860      *PI                          *
    0680   80000000  000001F0  00060382  3B000562  80000000  00000000  00060382  3B000A72      *                            *
    06A0   80000000  00000000  00060382  3B00015C  80000000  00000000  0007C00D  BA001000      *      0                     *
    06C0   00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000      *                            *
```

Figure 4-7 (Part 5 of 8). RPG Formatted Dump

```
RPG/400 FORMATTED DUMP                                                                                                    M
NAME     OFFSET  ATTRIBUTES    VALUE
..MDFDEV 0009B0  CHAR(10)      '          '                    '00000000000000000000'X
..MDFDVP 0009A0  POINTER(SPP)
                 SPACE OFFSET  2480                             '000009B0'X
                 OBJECT        PSSA
..MDFNDI 001990  CHAR(1)       '0'
.ACTPTR  001A60  POINTER(SYP)
                 CONTEXT       QGPL
                 OBJECT        SAMPLE
.ACTPTRC 001A60  CHAR(16)      '                '               '00000000000000000703AA08000238'X
.BINF1   000610  BINARY(4)     0
.BINF2   000614  BINARY(4)     0
.BINRF   000618  BINARY(4)     0
.BLANKS  0002B2  CHAR(140)     ' '
.BPCA    0004B0  CHAR(32767)   CANNOT DUMP - SPACE ADDRESSING OR BOUNDARY ALIGNMENT EXCEPTION
.BPCAPTR 000A00  POINTER(SPP)
                 SPACE OFFSET  1200                             '000004B0'X
                 OBJECT        FILEIN2   QGPL      FILEIN2
.BUFFER  0004D0  CHAR(148)     '                                                                                  '
         0004D0  VALUE IN HEX  '40404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040000010000'X
         0004FD    +46         '00010000000040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
         00052A    +91         '40404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
         000557    +136        '40404040404040404040404040'X
.BUFPTR  000980  POINTER(SPP)
                 SPACE OFFSET  1232                             '000004D0'X
                 OBJECT        FILEIN2   QGPL      FILEIN2
.CALLERR 001BF0  POINTER(IP)
                 INSTR #               14
                 CONTEXT       QGPL
                 OBJECT        SAMPLE
.CALLSW  001B2D  CHAR(1)       '1'
.CLOSASW 001B5A  CHAR(1)       '0'
.CLOSPTR 0000A0  POINTER(SYP)
                 CONTEXT       QSYS
                 OBJECT        QDMCLOSE
.C001001 001980  POINTER(SPP)
                 SPACE OFFSET  6480                             '00001950'X
                 OBJECT        PSSA
.C002002 0019C0  POINTER(SPP)
                 SPACE OFFSET  6545                             '00001991'X
                 OBJECT        PSSA
.CURROP  000598  CHAR(5)       'WRITE'
.DBFIND  0003A7  CHAR(2)       DIMENSION(4)
         0003A7    (1)         '0V'
         0003A9    (2)         '1V'
         0003AB    (3)         '2V'
         0003AD    (4)         'LR'
.DBFINX  0003A7  CHAR(8)       '0V1V2VLR'
.DBICNT  0003AF  BINARY(2)     4
.DEACTSW 001CC1  CHAR(1)       '0'
.DMCBF   000190  CHAR(2)       DIMENSION(256)
         000190    (1)         '  '                           '2000'X
         000214    (2)         '  '                           '0000'X
         000298    (3)         '  '                           '0005'X
         00031C    (4)         '  '                           '3070'X
         0003A0    (5)         '  '                           '8000'X
```

Figure 4-7 (Part 6 of 8). RPG Formatted Dump

```
RPG/400 FORMATTED DUMP
.U03LBLN 00169A  BINARY(2)     -75
.U03LIBN 00169C  CHAR(10)      '*LIBL     '
.U03NXTU 001660  POINTER(SPP)  NULL
.U030DPB 001610  POINTER(SPP)
                 SPACE OFFSET  0                              '00000000'X
                 OBJECT        QSYSPRT   QSYS      *N
.U030FBK 001640  POINTER(SPP)
                 SPACE OFFSET  176                            '000000B0'X
                 OBJECT        QSYSPRT   QSYS      *N
.U030VFL 001702  BINARY(2)     9
.U03PCB  001709  CHAR(393)     'SAMPLE    QGPL
         001768    +96         4 LINES OF BLANKS SUPPRESSED
         001709  VALUE IN HEX  'E2C1D4D7D3C540404040D8C7D7D340404040404000000000000000000000000000000000000000000000000000000000000000'X
         001736    +46         8 LINES OF ZEROES SUPPRESSED
.U03PCBL 001713  CHAR(10)      'QGPL      '
.U03PCBP 001709  CHAR(10)      'SAMPLE    '
.U03RECL 0016E0  BINARY(2)     1
.U03RLEN 0016E2  BINARY(2)     132
.U03SEQK 0016F0  CHAR(1)       ' '                          '00'X
.U03SIA  001670  POINTER(SPP)  NULL
.U03SQCK 0016EE  BINARY(2)     5
.WCBUIND 001B25  CHAR(8)       '        '
.WORK    000520  CHAR(120)     '  '
         000520  VALUE IN HEX  '00004040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
         00054D    +46         '40404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
         00057A    +91         '40404040404040404040404040404040404040404040404040404040404040'X
.WORKBC  00051C  CHAR(4)       '    '                       '00000000'X
.WORKB2  00051C  BINARY(2)     0
.WORKB3  00051E  BINARY(2)     0
.WORKB4  00051C  BINARY(4)     0
*IN      000344  CHAR(1)       DIMENSION(99)
         0003A4    (1-97)      '0'                          N
         0003A5    (98)        '1'
         0003A6    (99)        '0'
*INIT    00033E  CHAR(1)       '1'
*INLR    000343  CHAR(1)       '0'
*INOV    000340  CHAR(1)       '0'
*INXX    00033F  CHAR(1)       '1'
*IN1V    000341  CHAR(1)       '0'
*IN2V    000342  CHAR(1)       '0'
*IN98    0003A5  CHAR(1)       '1'
*IN99    0003A6  CHAR(1)       '0'
ARR      000441  CHAR(1)       DIMENSION(45)  O
         00046D    (1-45)      ' '
C.NUM    0003B1  CHAR(144)     '20 09 35 07 05 02 44 21 17 26 19 43 11 24 41 10 28 49 37 24 16 13 01 16 47 42 18 15 31 27 45 12'
         000410    +96         ' 04 03 29 48 39 23 14 08 32 40 06 46 30 22 34 38'
CURLIN   000516  ZONED(3,0)    '404040'X      P
FILE1    0004CF  CHAR(8)       '        '
FIL1DS   000BBC  CHAR(45)      'FILEIN1 1100011READ R*DETC   2500    FILEA   '      Q
FIL2DS   000FCC  CHAR(45)      'FILEIN2 1000000READ R*DETC   2900    FILEB   '
NUM      0003B1  CHAR(9)       DIMENSION(16)
         0003B1    (1)         '20 09 35 '
         0003BA    (2)         '07 05 02 '
         0003C3    (3)         '44 21 17 '
         0003CC    (4)         '26 19 43 '
         0003D5    (5)         '11 24 41 '
         0003DE    (6)         '10 28 49 '
         0003E7    (7)         '37 24 16 '
```

Figure 4-7 (Part 7 of 8). RPG Formatted Dump

```
RPG/400 FORMATTED DUMP
         0003F0    (8)         '13 01 16 '
         0003F9    (9)         '47 42 18 '
         000402    (10)        '15 31 27 '
         00040B    (11)        '45 12 04 '
         000414    (12)        '03 29 48 '
         00041D    (13)        '39 23 14 '
         000426    (14)        '08 32 40 '
         00042F    (15)        '06 46 30 '
         000438    (16)        '22 34 38 '
NUMX     000470  CHAR(9)        '20 09 35 '
OCCRDS   0004CF  CHAR(45)       '                                             '
OCCRDS.O 0004A2  CHAR(45)       DIMENSION(2)                                  <-- R
         0004A2    (1)         '                                   FILEB     '
         0004CF    (2)         '                                             '
OPCDE1   0004DF  CHAR(6)        '     '
PRNTDS   000510  CHAR(9)        ' 3 4     '
REC      0004CF  CHAR(45)       '                                             '
RECFM1   000BE1  CHAR(8)        'FILEA   '
RECFM2   000FF1  CHAR(8)        'FILEB   '
REC1     0004D7  CHAR(8)        '        '
RTNE1    0004EA  CHAR(8)        '        '
SKAFTR   000514  CHAR(2)        '  '
SKBEFR   000512  CHAR(2)        ' 4'
SPAFTR   000511  CHAR(1)        '3'
SPBEFR   000510  CHAR(1)        ' '
STAT     0004E5  CHAR(5)        '    '
UDATE    00061C  ZONED(6,0)     11588          <-- S
UDAY     00061E  ZONED(2,0)     15
UMONTH   00061C  ZONED(2,0)     1
UYEAR    000620  ZONED(2,0)     88
WORK.    00051C  CHAR(124)      '     '
         00051C  VALUE IN HEX   '0000000000004040404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
         000549    +46          '4040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
         000576    +91          '40404040404040404040404040404040404040404040404040404040404040404040404040404040404040'X
X        00046E  PACKED(3,0)    2
ZIGNDECD 001B51  CHAR(1)        '0'  T
U
-> ZPGMSTUS 000622  CHAR(400)    'SAMPLE    0000000000000000*DETC   000                                        QGPL          '
         000681    +96          '                                                                      1235          '
         0006E0    +191         '         QSYSPRT 012350PEN I*INIT   *INIT          E53       QPGMR     0008110115880115881111'
         00073F    +286         '43011588111125000 1QRPGSRC   QGPL      SAMPLE'
ZZ01BIN  0004B0  CHAR(45)       '                         ¢                   '
         0004B0  VALUE IN HEX   '8000000000000000007C00DBD0008800000004A0037002D4040404040404040404040404040404040404040404040'X
ZZ01BOUT 000A30  CHAR(45)       '                                             '
         000A30  VALUE IN HEX   '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
   V
ZZ02BIN  0004D0  CHAR(41)       '                                       '
ZZ02BOUT 000A30  CHAR(41)       '                                       '
         000A30  VALUE IN HEX   '00000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
ZZ03BIN  001709  CHAR(132)      'SAMPLE    QGPL                                                                          '
         001768    +96          '                                   '
         001709  VALUE IN HEX   'E2C1D4D7D3C540404040D8C7D7D3404040404040000000000000000000000000000000000000000000000000000000000000'X
         001736    +46          2 LINES OF ZEROES SUPPRESSED
W
   X
ZZ03BOUT 001020  CHAR(132)      '                                                                                        '
         00107F    +96          '                                   '
         001020  VALUE IN HEX   '00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
         00104D    +46          2 LINES OF ZEROES SUPPRESSED
STATIC STORAGE FOR PROGRAM SAMPLE     BEGINS AT OFFSET 000290 IN THE PROGRAM STATIC STORAGE AREA (PSSA)
AUTOMATIC STORAGE FOR PROGRAM SAMPLE     BEGINS AT OFFSET 0015B0 IN THE PROGRAM AUTOMATIC STORAGE AREA (PASA)
RPG/400 FORMATTED DUMP
        * * * * *  E N D   O F   R P G   D U M P  * * * * *
```

Figure 4-7 (Part 8 of 8). RPG Formatted Dump

A Qualified program name and library.

B Current status code.

C Previous status code.

D RPG/400 source statement in error.

E RPG/400 routine in which the exception/error occurred.

F CPF or MCH for a machine exception.

G Machine instruction number.

H Information about the last file used in the program before an exception/error (RPG-1235) occurred.

I Program information.

J Error in the file.

K The number of times RPG/400 has requested I/O of the system (not the number of I/O operations requested by the program).

L The open data path is included in the dump if the user responds to an RPG/400 run-time message with an F option.

M A list of compiler-generated fields is also included in the dump if the user responds to an RPG/400 run-time message with an F option and if the program was compiled with a 1 in position 15 of the control specification.

N General indicators 1-99 and their current status (1 is on, 0 is off).

O Beginning of user fields.

P Invalid zoned field printed in hexadecimal.

Q File information data structures for FILEIN1 and FILEIN2.

R Double-occurrence data structure.

S System data values.

T IGNDECERR(*NO) was specified in the CRTRPGPGM command.

U Program status data area.

V Input buffer for file 02.

W This is the file number. See the cross-reference section of the compiler listing for the corresponding file name. The files are assigned a sequence number corresponding to the order in which they are defined in your specifications. Thus, the file number 03 corresponds to the FILEIN2 file described in this program.

X Output buffer for file 03.

File Information Data Structure

A file information data structure (INFDS) is always provided and updated for each file in an RPG/400 program. "Exception/Error Handling" on page 4-35 describes how you can define this data structure and how you can reference its subfields.

Even if a file information data structure is not specified, the information in it is available for program debugging through the OS/400 test facilities and through the RPG/400 DUMP operation.

Table 4-1 to Table 4-5 provide the layout of the subfields of the feedback information available in the file information data structure. The predefined From and To positions can be used to access the feedback information.

The input/output feedback information section (positions 241 through 366) and the device-dependent feedback information section (positions 367 through 528) of the file information data structure are not updated for each operation to files in which the records are blocked and unblocked by RPG/400. The feedback information is updated only when a block of records is transferred between RPG/400 and OS/400. For a potential run time performance improvement of input and output operations, RPG/400 unblocks input records and blocks output records in SEQ or DISK files if:

- The file is an output-only file (O is specified in position 15 of the file description specifications) and contains only one record format if the file is externally described.
- The file is a combined table file (C is specified in position 15 and T in position 16 of the file description specifications).
- The file is an input-only file (I is specified in position 15 of the file description specifications), contains only one record format if the file is externally described, and uses only the OPEN, CLOSE, FEOD, and READ operation codes.

Even if all of the above conditions are met, certain OS/400 restrictions may prevent blocking and unblocking. In these cases, performance is not improved and the input/output feedback area is updated for each input/output operation.

You can obtain valid updated feedback information by using the CL command OVRDBF (Override with Data Base File) with SEQONLY(*NO) specified. If you use a file override command, RPG/400 does not block or unblock the records in the file.

Table 4-1 (Page 1 of 2). Contents of the File Feedback Information available in the File Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
1	8	Character	8	File name (same as subfield location *FILE).
9	9	Character	1	Open indication (1 = open).
10	10	Character	1	End of file (1 = end of file)

Table 4-1 (Page 2 of 2). Contents of the File Feedback Information available in the File Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
11	15	Zoned decimal	5 (zero decimal positions)	Status code (same as subfield location *STATUS).
16	21	Character	6	Operation code (same as subfield location *OPCODE).
22	29	Character	8	Name of the RPG/400 routine in which the exception/error occurred (same as subfield location *ROUTINE).
30	37	Character	8	RPG/400 source statement sequence number.
38	42	Zoned decimal	5 (zero decimal positions)	User-specified reason for error on SPECIAL file.
38	45	Character	8	For an externally described file, the RPG/400 name of the record being processed when the exception/error occurred. For a program described file, the record identifying indicator is placed left-adjusted in the field (same as the subfield location *RECORD).
46	52	Character	7	Exception number (machine or CPF message number).
53	56	Character	4	MI/ODT (machine instruction/object definition template) number.
57	66	Character	10	Unused.

Table 4-2 (Page 1 of 2). Contents of the File Feedback Information available in the File Information Data Structure (INFDS) Valid after a POST Operation to a Specific Device

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
67	70	Zoned decimal	4 (zero decimal positions)	Screen size (same as subfield location *SIZE).
71	72	Zoned decimal	2 (zero decimal positions)	The national language input capability of the device is for single characters. The value is 0 (same as subfield location *INP).

Table 4-2 (Page 2 of 2). Contents of the File Feedback Information available in the File Information Data Structure (INFDS) Valid after a POST Operation to a Specific Device

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
73	74	Zoned decimal	2 (zero decimal positions)	The national language output capability of the device is for single characters. The value is 0 (same as subfield location *OUT).
75	76	Zoned decimal	2 (zero decimal positions)	The preferred national language mode of the device is for single characters. The value is 0 (same as subfield location *MODE).

Note: The remaining feedback information (positions 81 through 528) is copied from the open feedback and I/O feedback areas, which are described in the *Programming: Data Management Guide*. The description in the information column indicates system usage of the fields and may not apply to the RPG/400 user. For example, information in positions 241 and 242 is used by the system to determine the start of the file-dependent feedback area and is not applicable to the RPG/400 user.

Table 4-3 (Page 1 of 4). Contents of the Open Feedback Information available in the File Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
81	82	Character	2	Open, data path (ODP) type: DS Device file DB Data base member SP Spooled file
83	92	Character	10	Name of the file. For a nonspooled file, this is the name of the file actually being opened. For a spooled file, this is the name of the device file or the inline data file being opened.
93	102	Character	10	Name of the library containing the file. For a spooled input file, this is *N.
103	112	Character	10	Name of the spooled file. This entry is set only if the ODP type is SP. It is the name of a data base file containing the spooled input or output records.
113	122	Character	10	Name of the library in which the spooled file is located.

Table 4-3 (Page 2 of 4). Contents of the Open Feedback Information available in the File Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
123	124	Binary	2	Spooled file number (supplied only for spooled output).
125	126	Binary	2	Record length (number of bytes transferred at a time).
127	128	Binary	2	Reserved.
129	138	Character	10	Member name: • If ODP type is DB, this entry is the member name in the file named in positions 83 through 92. • If ODP type is SP, this entry is the member name in the file named in positions 103 through 112.
139	142	Binary	4	Not used.
143	146	Binary	4	Not used.
147	148	Binary	2	File type (supplied only if the ODP type is DS or SP). See *Programming: Data Management Guide*, for file type codes.
149	151	Character	3	Reserved.
152	153	Binary	2	Number of rows on a display screen or number of lines on a printed page (supplied only for a display device or a printer).
154	155	Binary	2	Number of columns on a display screen or number of characters per printed line (supplied only for display device or printer).
156	159	Binary	4	Number of records in the member at open time. This entry is supplied only if the ODP type is DB or SP and the file is being opened for input.

Table 4-3 (Page 3 of 4). Contents of the Open Feedback Information available in the File Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
160	161	Character	2	Access type (supplied only if ODP type is DB): KU Keyed, unique KF Keyed, first-in-first-out (FIFO) with duplicate keys KL Keyed, last-in-first-out (LIFO) with duplicate keys AR Arrival sequence
162	162	Character	1	Duplicate key indication. This entry is set only if the access path is KU, KF, or KL:. D Duplicate keys allowed if the access path is KF or KL. U Duplicate keys are not allowed; all keys are unique and the access path is KU.
163	163	Character	1	Source file indication. This entry contains Y if this file is being opened as a source file.
164	173	Character	10	User file control block (UFCB) parameters. This entry indicates which UFCB parameters are in effect.
174	183	Character	10	User file control block (UFCB) overrides. This entry indicates which override parameters are in effect.
184	185	Binary	2	Offset to volume label fields of open feedback area. This entry is supplied only for tape or diskette; otherwise, the field contains zero.
186	187	Binary	2	Maximum number of records that can be sent or received in a block when using blocked record I/O.
188	189	Binary	2	Overflow line number (supplied only for a printer file).

Table 4-3 (Page 4 of 4). Contents of the Open Feedback Information available in the File Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
190	191	Binary	2	Blocked record I/O record increment. This is the number of bytes to add to the address of a record to get the address of the next record in a block.
192	196		5	Unused.
197	206	Character	10	Name of the requester program device.
207	240		34	Unused.

See Table 4-5 on page 4-31 for the values of positions 241 to 528 after a POST operation with a device specified in factor 1.

Table 4-4 (Page 1 of 3). Contents of the Input/Output Feedback Information available in the File Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
241	242	Binary	2	Offset to file-dependent feedback information. See the *Data Management Guide* for the layout of feedback information for specific files.
243	246	Binary	4	Write operation count. This entry is updated only when a Write operation completes successfully. For information specific to ICF, see the corresponding entry in the *Communications: Programmer's Guide.*
247	250	Binary	4	Read operation count. This entry is updated only when a Read operation completes successfully. For data base files, this entry is not updated for a position-only request. For information specific to ICF , see the corresponding entry in the *Communications: Programmer's Guide.*
251	254	Binary	4	Write/Read operation count. This entry is updated only when a Write/Read operation completes successfully.

Table 4-4 (Page 2 of 3). Contents of the Input/Output Feedback Information available in the File Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
255	258	Binary	4	Other I/O operation count. Number of successful operations other than Write, Read, or Write/Read. Updated only when the operation completes successfully. This count includes update, delete, force-end-of-data, force-end-of-volume, ACQ, REL, and release record lock requests.
259	259		1	Unused.
260	260	Character	1	Current operation. This entry represents the last operation requested. See *Data Management Guide*, Common I/O Feedback Area table for operation codes.
261	270	Character	10	Name of the record format just processed, which is either: • Specified on the I/O request, or • Determined by system processing For a display device, the default name is either the name of the only record format in the file or the previous record format name for the record put to the display that contains input data. For ICF files, see the description of the FMTSLT parameter on the CL commands ADDICFDEVE and OVRICFDEVE in the *Communications: Programmer's Guide*.
271	272	Character	2	Device class. See *Data Management Guide*, Common I/O Feedback Area table for device class codes.
273	282	Character	10	Program device name. This entry is the name of the program device for which the operation just completed.

Table 4-4 (Page 3 of 3). Contents of the Input/Output Feedback Information available in the File Information Data Structure (INFDS)

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
283	286	Binary	4	Length of the record processed by the last I/O operation (supplied for display device file, data base file, tape file, and ICF file). On ICF write operations, this entry is the record length of the data. On ICF read operations, this entry is the length of the record for the last get operation.

File-Dependent Feedback Information.

Complete file-dependent feedback information is available in the *Programming: Data Management Guide*. Refer to the section "I/O Feedback Area" and the tables on the file-dependent area in that section.

To calculate the From and To positions (positions 44 through 47 and 48 through 51 of the input specifications that specify the subfields of the file information data structure (INFDS) for the file-dependent area, use the Offset, Data Type, and Length given in the *Programming: Data Management Guide* and do the following calculations:

```
From = 367 + Offset
To = From - 1 + Character-Length
Character-Length = Length, if Data Type is Character, or
                 = (Length +1)/8, if Data Type is Binary
```

For example, for the relative record number of a subfile record, the *Programming: Data Management Guide* gives

```
Offset = 9, Data Type is Binary,
Length = 15.
  Therefore,
From = 367 + 9 = 376,
Character-Length = (15 +1)/8 = 2, and
To = 376-1 + 2 = 377.
```

Contents of File Information Data Structure After POST

After a POST operation with a program device specified in factor 1, the following information overlays positions 241 to 528.

Table 4-5 (Page 1 of 3). Contents of the Input/Output Feedback Information available in the File Information Data Structure (INFDS) after a POST Operation

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
241	250	Character	10	Program device name.
251	260	Character	10	Device description name.
261	270	Character	10	User ID
271	271	Character	1	Device class: • 'D' is display • 'I' is ICF • 'U' is unknown
272	277	Character	6	Device type: • '3179' 3179 display • '317902' 3179 model 2 display • '3180' 3180 display • '3196A' 3196 model A1/A2 display • '3196B' 3196 model B1/B2 display • '3197C1' 3197 model C1 display • '3197C2' 3197 model C2 display • '3197D1' 3197 model D1 display • '3197D2' 3197 model D2 display • '3197W1' 3197 model W1 display • '3197W2' 3197 model W2 display • '3270' 3270 display • '525111' 5251 display • '5291' 5291 display • '5292' 5292 display • '529202' 5292 model 2 display • '5555B1' 5555 model B01 display • '5555E1' 5555 model E01 display • 'BSCEL' BSCEL • 'APPC' APPC • 'ASYNC' ASYNC • 'SNUF' SNUF
278	278	Character	1	*REQUESTER device. This flag indicates whether this entry is defining a *REQUESTER device. • 'Y' is requesting program device • 'N' is not a requesting program device

Table 4-5 (Page 2 of 3). Contents of the Input/Output Feedback Information available in the File Information Data Structure (INFDS) after a POST Operation

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
279	279	Character	1	Acquire status. Set even if a device is implicitly acquired at open time. • 'Y' device is acquired • 'N' device is not acquired
280	280	Character	1	Invite status. • 'Y' device is invited • 'N' device is not invited
281	281	Character	1	Data available • 'Y' invited data is available • 'N' no data is available
282	283	Binary	2	Number of rows on display
284	285	Binary	2	Number of columns on display
286	286	Character	1	Display allow blink capability • 'Y' blink capable • 'N' not blink capable
287	287	Character	1	Online/Offline status • 'O' display is online • 'F' display is offline
288	288	Character	1	Display location • 'L' local display • 'R' remote display
289	289	Character	1	Display type • 'A' Alphanumeric or Katakana • 'I' Ideographic
290	290	Character	1	Keyboard type of display • 'A' Alphanumeric or Katakana keyboard • 'I' Ideographic keyboard
291	291	Character	2	Session status (All communication types) • 'N' transaction is not started • 'Y' transaction is started

Table 4-5 (Page 3 of 3). Contents of the Input/Output Feedback Information available in the File Information Data Structure (INFDS) after a POST Operation

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
292	292	Character	1	Synchronization level (APPC communications only) • '0' synchronization level 0 (SYNCLVL(*NONE)) • '1' synchronization level 1 (SYNCLVL(*CONFIRM))
293	293	Character	1	Conversation being used (APPC communications only) • 'M' mapped conversation • 'B' basic conversation
294	301	Character	8	Remote Location (All communications types)
302	309	Character	8	Local LU name (APPC communications only)
310	317	Character	8	Local network ID (APPC communications only)
318	325	Character	8	Remote LU name (APPC communications only)
326	333	Character	8	Remote network ID (APPC communications only)
334	341	Character	8	Mode (APPC communications type only)
342	528	Character	43	Reserved fields

Program Status Data Structure

A program status data structure can be defined to make program exception/error information available to an RPG/400 program. "Exception/Error Handling" on page 4-35 describes how you can define this data structure and how you can reference its subfields.

Table 4-6 on page 4-34 provides the layout of the subfields of the data structure and the predefined From and To positions of its subfields, which can be used to access information in this data structure.

Table 4-6 (Page 1 of 2). Contents of the Program Status Data Structure

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
1	10	Character	10	Program name (same as subfield location *PROGRAM).
11	15	Zoned decimal	5 (zero decimal positions)	Status code (same as subfield location *STATUS).
16	20	Zoned decimal	5 (zero decimal positions)	Previous status code.
21	28	Character	8	RPG/400 source statement sequence number.
29	36	Character	8	Name of the RPG/400 routine in which the exception/error occurred (same as subfield location *ROUTINE).
37	39	Zoned decimal	3 (zero decimal positions)	Number of parameters passed to this program (same as subfield location *PARMS).
40	42	Character	3	Exception type (CPF for a OS/400 exception or MCH for a machine exception).
43	46	Character	4	Exception number. For a CPF exception, this field contains a CPF message number. For a machine exception, it contains a machine exception number.
47	50	Character	4	MI/ODT (machine instruction/object definition template) number.
51	80	Character	30	Work area for messages.
81	90	Character	10	Name of library in which the program is located.
91	170	Character	80	Retrieved exception data. CPF messages are placed in this subfield when location *STATUS contains 09999.
171	174	Character	4	Identification of the exception that caused RPG9001 exception to be signaled (the called program failed).
175	200		26	Unused.
201	208	Character	8	Name of file on which the last file operation occurred (updated only when an error occurs).

Table 4-6 (Page 2 of 2). Contents of the Program Status Data Structure

From (Positions 44-47)	To (Positions 48-51)	Format	Length	Information
209	243	Character	35	Status information on the last file used. This information includes the status code, the RPG/400 routine name, the statement number, and record name. It is updated only when an error occurs.
244	253	Character	10	Job name.
254	263	Character	10	User name from the user profile.
264	269	Zoned decimal	6 (zero decimal positions)	Job number.
270	275	Zoned decimal	6 (zero decimal positions)	Date (in UDATE format) the job was entered in the system (UDATE is derived from this date).
276	281	Zoned decimal	6 (zero decimal positions)	Date of program running (the system date in UDATE format).
282	287	Zoned decimal	6 (zero decimal positions)	Time of program running in the format hhmmss.
288	293	Character	6	Date (in UDATE format) on which the program was compiled.
294	299	Character	6	Time (in the format hhmmss) the program was compiled.
300	303	Character	4	Level of the compiler.
304	313	Character	10	Source file name.
314	323	Character	10	Source library name.
324	333	Character	10	Source file member name.
334	429		96	Unused.

Exception/Error Handling

RPG/400 handles two types of exception/errors: program exception/errors and file exception/errors. Some examples of program exception/errors are division by zero, invalid array index, or SQRT of a negative number. Some examples of file exception/errors are undefined record type or a device error.

You can choose to let the RPG/400 default error handler receive control when an exception/error occurs, or you can choose to control the program logic yourself when an exception/error occurs. To control the program logic, you can:

- Code a file exception/error subroutine, which is defined by the INFSR keyword on a file description specifications continuation line, to handle file exception/errors.
- Code a program exception/error subroutine, which is referred to as *PSSR, to handle the program exception/errors.
- Use an indicator in positions 56 and 57 of the calculation specifications for a file operation code to control the logic if an exception/error occurs on that file operation code.
- Specify a program error indicator on other calculation operations (such as OCUR or CALL).

You can specify a file information data structure, which is defined by the INFDS keyword on a file description specifications continuation line, and a program status data structure, which is identified by an S in position 18 of the data structure statement on the input specifications. You must include status information on the file or program exception/error that occurred. You can determine which file or program exception/errors you want to handle in the associated subroutine. You can also determine the exception/error you want the RPG/400 default error handler to control. You can do this using the information in the data structures.

Whenever an exception/error occurs, RPG/400 checks for the presence of an exception/error indicator on a calculation operation or the presence of the appropriate subroutine. If these are not present, the exception/error is handled by the RPG/400 default error handler. Figure 4-8 on page 4-37 shows the specific steps in the logic flow if an exception/error occurs.

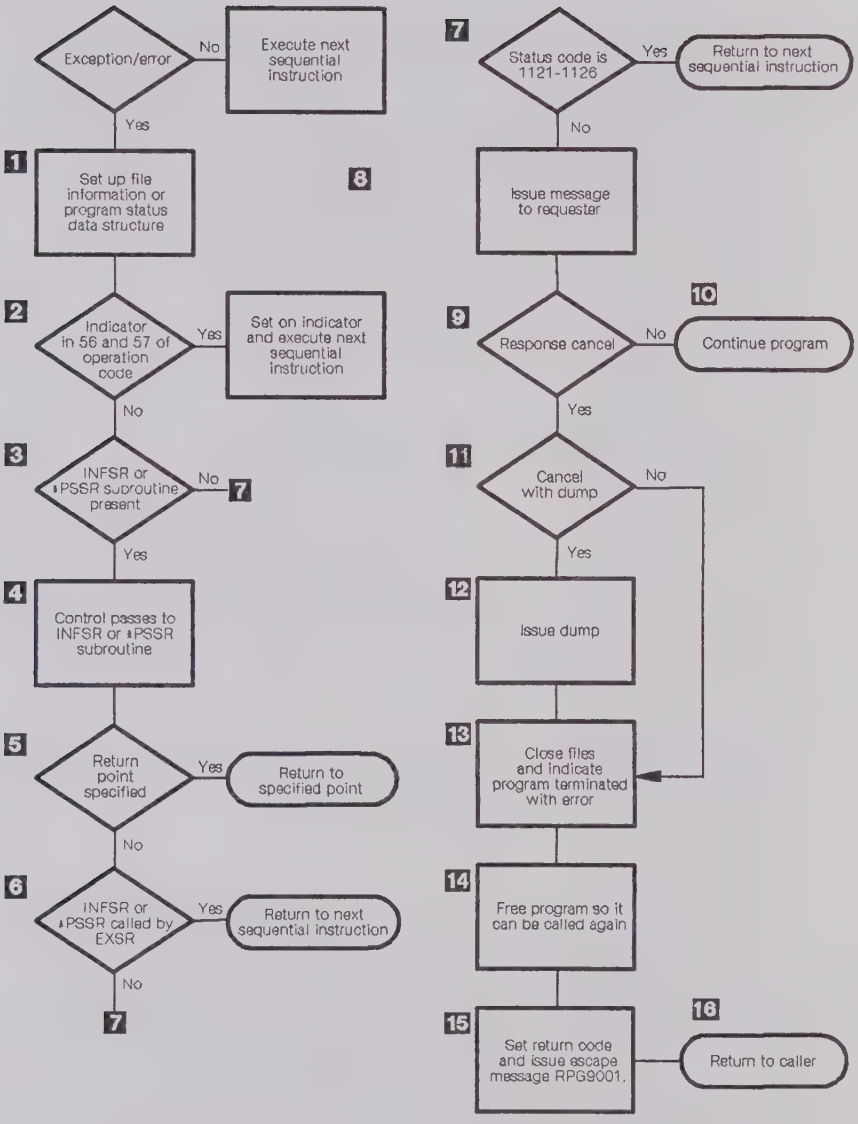


Figure 4-8. Logic Flow for Exception/Error Handling (Repeats Figure 1-4)

The following numbers refer to the numbers in Figure 4-8:

1. Set up the file information or program status data structure, if specified, with status information.
2. If the exception/error occurred on an operation code that has an indicator specified in positions 56 and 57, RPG/400 sets the indicator, and control returns to the next sequential instruction in the calculations.
3. RPG/400 determines whether the appropriate exception/error subroutine (INFSR or *PSSR) is present in the program. If it is, the program goes to step 4; if not, the program branches to step 7.

4. Control passes to the exception/error subroutine (INFSR or *PSSR).
5. RPG/400 determines whether a return point is specified in factor 2 of the ENDSR operation for the exception/error subroutine. If a return point is specified, the program goes to the specified return point. If a return point is not specified, the program goes to step 6. If a field name is specified in factor 2 of the ENDSR operation and the content is not one of the RPG/400-defined return points (such as *GETIN or *DETC), the program goes to step 6. No error is indicated, and the original error is handled as though the factor 2 entry were blank.
6. If the exception/error subroutine was called explicitly by the EXSR operation, the program returns to the next sequential instruction. If not, the program continues with step 7.
7. If the Status code is 1121-1126 (see"Status Codes" on page 4-42) control returns to the next sequential instruction in the calculations.
8. A message is issued to the requestor. For an interactive job, the message goes to the requester. For a batch job, the message goes to QSYSOPR; if QSYSOPR is not in break mode, a default response is issued.
9. RPG/400 determines whether the user's response is to cancel the program. If the response is to cancel, the program branches to step 10. If not, the program continues with step 9.
10. The program continues processing at *GETIN.
11. RPG/400 determines whether the user's response is to cancel with a dump. If the response is to cancel with a dump, the program continues with step 11. If not, the program branches to step 12.
12. RPG/400 issues a dump.
13. RPG/400 closes all files and sets the return code to indicate that the program ended with an error.
14. RPG/400 frees the program so that it can be called again, and the program returns to the caller.
15. Set return code and issue escape message RPG9001.
16. Return to caller.

File Exception/Errors

Examples of exception/errors occurring in a file operation are:

- Undefined record type
- No record found
- Match field sequence
- Permanent I/O error.

File exception/errors can be handled in one of the following three ways:

- An indicator can be specified in positions 56 and 57 of the calculation specifications for a file operation code. This indicator is set on if an exception/error

occurs during the processing of the specified file operation. The optional file information data structure (INFDS) is updated with the exception/error information. You determine the action to be taken by testing the indicator.

- The file exception/error subroutine (INFSR), that you write can be specified for the file. If an indicator is not specified in positions 56 and 57 for the file operation, the INFSR subroutine receives control when an exception/error occurs on the file operation. You can test the information in the optional file information data structure (INFDS) to determine the action to be taken in the subroutine.
- If neither the error indicators nor the INFSR subroutine are specified in the program, the RPG/400 default error handler receives control and for exception/error Status codes other than 1122-1126 (see "Status Codes" on page 4-42) a message is issued.
- A return point can be specified on the ENDSR statement of the exception/error subroutine when the program is to continue running. If the entry is invalid, the RPG/400 default error handler receives control.

Exception/Error Indicators

An indicator can be specified in positions 56 and 57 of the calculation specifications for a file operation code. If an exception/error occurs on the file operation, the indicator is set on, the optional file information data structure is updated, and control returns to the next sequential instruction. You can then test the indicator to determine what action to take.

When an indicator is specified on a file operation, you can explicitly call the file exception/error subroutine (INFSR) with the EXSR operation. If the INFSR is explicitly called by the EXSR operation and factor 2 of the ENDSR operation is blank or the field specified has a value of blank, control returns to the next sequential instruction.

File Information Data Structure (INFDS)

A file information data structure (INFDS) can be defined for each file to make file exception/error information available to the program. The file information data structure must be unique for each file. A file information data structure contains predefined subfields that identify:

- The name of the file for which the exception/error occurred
- The record being processed when the exception/error occurred or the record that caused the exception/error
- The last operation being processed when the exception/error occurred
- The status code
- The RPG/400 routine in which the exception/error occurred.

Note: The file information data structure is always provided and updated even if INFDS is not specified in the program. If INFDS is not specified, the information in the data structure is available for program debugging through the OS/400 test facilities or through the DUMP operation. For an example of the information that is provided with the DUMP operation, see "Using the RPG/400 Formatted Dump" on page 4-13. The *Programming: Control Language Programmer's Guide* describes the interactive debug feature in detail.

You can use the file information data structure to access the information in the data management feedback area, which is available following each I/O operation. For information on how to use predefined From and To positions to access the feedback information, see "File Information Data Structure" on page 4-23.

To specify a file information data structure, make the following entries on a continuation line (or on the main file specification line) for the file description specifications:

Position	Entry
6	F
7-52	Blank (if the information is specified on a separate continuation line)
53	K (indicates a continuation statement)
54-59	INFDS (identifies this data structure as a file information data structure)
60-65	Name of the file information data structure.

Also make the following entries on an input specification line:

Position	Entry
6	I
7-12	Name of the file information data structure
13-18	Blank
19-20	DS
21-74	Blank.

For each subfield of the information data structure, make the following entries on an input specification line:

Position	Entry
6	I
7-43	Blank
44-51	Either (1) a special keyword (listed below), or (2) a From position in the file information data structure (in positions 44 through 47) and a To position in the information data structure (in positions 48 through 51)
52	Blank
53-58	Name of the subfield of the information data structure
59-74	Blank.

The location of the subfields in the file information data structure is defined by special keywords or by predefined From and To positions. For specific values for the From and To positions in the information data structure, see "File Information Data Structure" on page 4-23.

Specify the special keywords left-adjusted, in positions 44 through 51. The keywords are not labels and cannot be used to access the subfields. Short entries are padded on the right with blanks. The keywords and their descriptions are as follows:

Keyword	Description
*FILE	Eight-position character field that identifies the name of the file (as specified in positions 7 through 14 of the file description specifications).
*INP	A two-digit numeric field containing 0. The national language input capability of the device is for single characters.
*MODE	A two-digit numeric field containing 0. The preferred national language mode of the device is for single characters.
*OUT	A two-digit numeric field containing 0. The national language output capability of the device is for single characters.
*OPCODE	Six-position character field that contains the name of the last operation processed on the file when the exception/error occurred. The first five positions (left-adjusted) specify the type of operation by using the character representation of the calculation operation codes. For example, if a READE was being processed when an exception/error occurred, READE is placed in the leftmost five positions. If the operation was an implicit operation processed by RPG/400 (for example, a primary file read or update on the output specifications), the equivalent operation code is generated by RPG/400 (such as READ or UPDAT) and placed in location *OPCODE. The remaining position contains one of the following: F The last operation was specified for a file name. R The last operation was specified for a record. I The last operation was an implicit file operation. The field is blank after a POST operation for a specific device.
*SIZE	A four-digit numeric field containing the product of the number of rows and the number of columns on the device screen.
*STATUS	Five-digit numeric field, with zero decimal positions, that contains the status code. For a description of these codes, see "Status Codes" in the following text.

Keyword	Description
*RECORD	*Externally described file:* Eight-position character field that contains the RPG/400 name of the record being processed when the exception/error occurred. *Program described file:* Eight-position character field in which the record identifying indicator is placed left-adjusted; the remaining six positions are filled with blanks. The field is blank after a POST operation for a specific device.
*ROUTINE	Eight-position character field that contains the name of the RPG/400 routine in which the exception/error occurred. This subfield is updated at the beginning of an RPG/400 routine or after a program call only when the location *STATUS is updated with a value of nonzero. The following names identify the routines: *INIT Program initialization *DETL Detail lines *GETIN Get input record *TOTC Total calculations *TOTL Total lines *DETC Detail calculations *OFL Overflow lines *TERM Program ending pgmname Name of program called (first 8 characters).

Status Codes

Any code placed in the subfield location *STATUS that is greater than 99 is considered to be an exception/error condition. If the status code is greater than 99, the error indicator, if specified in positions 56 and 57, is set on or the file exception/error subroutine receives control. Location *STATUS is updated after every file operation.

The following codes are placed in the subfield location *STATUS for the file information data structure:

Normal Codes

Code	Device[1]	RC[2]	Condition
00000			No exception/error
00002	W	n/a	Function key used to end display
00011	W,D,SQ	11xx	End of file on a read (input)

Code	Device[1]	RC[2]	Condition
00012	W,D,SQ	n/a	No-record-found condition on a CHAIN operation
00013	W	n/a	Subfile is full.

Exception/Error Conditions

Code	Device[1]	RC[2]	Condition
01011	W,D,SQ	n/a	Undefined record type (input record does not match record identifying indicator)
01021	W,D,SQ	n/a	Tried to write a record that already exists (file being used has unique keys and key is duplicate, or attempted to write duplicate relative record number to a subfile)
01031	W,D,SQ	n/a	Match field out of sequence
01041	n/a	n/a	Array/table load sequence error
01051	n/a	n/a	Excess entries in array/table file
01052	n/a	n/a	Clearing of table prior to dump of data failed
01071	W,D,SQ	n/a	Numeric sequence error
01121[4]	W	n/a	No indicator on the DDS keyword for Print key
01122[4]	W	n/a	No indicator on the DDS keyword for Roll Up key
01123[4]	W	n/a	No indicator on the DDS keyword for Roll Down key
01124[4]	W	n/a	No indicator on the DDS keyword for Clear key
01125[4]	W	n/a	No indicator on the DDS keyword for Help key
01126[4]	W	n/a	No indicator on the DDS keyword for Home key
01201	W	34xx	Record mismatch detected on input
01211	all	n/a	I/O operation to a closed file
01215	all	n/a	OPEN issued to a file already opened
01216[3]	all	yes	Error on an implicit OPEN/CLOSE operation
01217[3]	all	yes	Error on an explicit OPEN/CLOSE operation
01218	D,SQ	n/a	Unable to allocate record
01221	D,SQ	n/a	Update operation attempted without a prior read
01231	SP	n/a	Error on SPECIAL file
01235	P	n/a	Error in PRTCTL space or skip entries
01241	D,SQ	n/a	Record number not found (record number specified in record address file is not present in file being processed.)

Code	Device[1]	RC[2]	Condition
01251	W	80xx 81xx	Permanent I/O error occurred
01255	W	82xx 83xx	Session or device error occurred. Recovery may be possible.
01261	W	n/a	Attempt to exceed maximum number of acquired devices
01281	W	n/a	Operation to unacquired device.
01282	W	0309	Job ending with controlled option
01285	W	n/a	Attempt to acquire an already acquired device
01286	W	n/a	Attempt to open shared file with SAVDS or IND options
01287	W	n/a	Response indicators overlap IND indicators
01299	W,D,SQ	yes	Other I/O error detected
01331	W	0310	Wait time exceeded for READ from WORKSTN file

[1]In the preceding tables device refers to the devices for which the condition applies. The following abbreviations are used:

P = PRINTER
D = DISK
W = WORKSTN
SP = SPECIAL
SQ = Sequential

The major/minor return codes under column RC apply only to WORKSTN files.

[2]The formula mmnn is used to described major/minor return codes: mm is the major and nn the minor.

[3]Any errors that occur during an open or close operation will result in a *STATUS value of 1216 or 1217 regardless of the major/minor return code value.

[4]See Figure 4-8 on page 4-37 for special handling.

Note

For System/38 Environment *STATUS see *RPG III Reference Manual and Programmer's Guide*, SC21-7725.

The following table shows the major/minor return code to *STATUS value mapping for errors that occur to AS/400 programs using WORKSTN files only.

Major	Minor	*STATUS
00,02	all	00000

Major	Minor	*STATUS
03	all (except 09,10)	00000
03	09	01282
03	10	01331
04	all	01299
08	all	01285[1]
11	all	00011
34	all	01201
80,81	all	01251
82,83	all	01255

[1]The return code field will not be updated for a *STATUS value of 1285, 1261, or 1281 because RPG detects these conditions before calling data management. To monitor for these errors you must check for the *STATUS value and not for the corresponding major/minor return code value.

File Exception/Error Subroutine (INFSR)

To identify the user-written RPG/400 subroutine that may receive control following file exception/errors, make the following entries on a continuation line (or on the main file line) for the file description specifications:

Position	Entry
6	F
7-52	Blank (if the information is specified on a separate continuation line)
53	K (indicates a continuation statement)
54-59	INFSR
60-65	Name of the BEGSR subroutine that receives control when exception/errors occur on this file. The subroutine name can be *PSSR, which indicates that the program exception/error subroutine is given control for the exception/errors on this file.

A file exception/error subroutine (INFSR) receives control when an exception/error occurs on an implicit (primary or secondary) file operation or on an explicit file operation that does not have an indicator specified in positions 56 and 57. The file exception/error subroutine can also be run by the EXSR operation code. Any of the RPG/400 operations can be used in the file exception/error subroutine. Factor 1 of the BEGSR operation and factor 2 of the EXSR operation must contain the name of the subroutine that receives control (same name as specified in positions 60 through 65 of the file description specifications continuation line). The ENDSR operation must be the last specification for the file exception/error subroutine and should be specified as follows:

Position	Entry
6	C
7-17	Blank.
18-27	Can contain a label that is used in a GOTO specification within the subroutine.
28-32	ENDSR.
33-42	Optional entry to designate where control is to be returned following processing of the subroutine. The entry must be a 6-position character field, literal, or array element whose value specifies one of the following return points. **Note:** If the return points are specified as literals, they must be enclosed in apostrophes. If they are specified in fields or array elements, the value must be left-adjusted in the field or array element. *DETL — Continue at the beginning of detail lines. *GETIN — Continue at the get input record routine. *TOTC — Continue at the beginning of total calculations. *TOTL — Continue at the beginning of total lines. *OFL — Continue at the beginning of overflow lines. *DETC — Continue at the beginning of detail calculations. *CANCL — Cancel the processing of the program (see "An Abnormal End" on page 11-12 in Chapter 11, "Communicating with Objects in the System"). Blanks — Return control to the RPG/400 default error handler. This applies to both when factor 2 is a value of blanks and when factor 2 is not specified. If the subroutine was called by the EXSR operation and factor 2 is blank, control returns to the next sequential instruction.
43-59	Blank.

Remember the following when specifying the file exception/error subroutine:

- The programmer can explicitly call the file exception/error subroutine by specifying the name of the subroutine in factor 2 of the EXSR operation.
- After the ENDSR operation of the file exception/error subroutine is run, RPG/400 resets the field or array element specified in factor 2 to blanks. Thus, if the programmer does not place a value in this field during the processing of the subroutine, the RPG/400 default error handler receives control following processing of the subroutine unless the subroutine was called by the EXSR operation. Because factor 2 is set to blanks, the programmer can specify the return point

within the subroutine that is best suited for the exception/error that occurred. If the subroutine was called by the EXSR operation and factor 2 of the ENDSR operation is blank, control returns to the next sequential instruction following the EXSR operation. A file exception/error subroutine can handle errors in more than one file.

- If a file exception/error occurs during the start or end of a program, control passes to the RPG/400 default error handler, and not to the user-written file exception/error or subroutine (INFSR).
- Because the file exception/error subroutine may receive control whenever a file exception/error occurs, an exception/error could occur while the subroutine is running if an I/O operation is processed on the file in error. If an exception/error occurs on the file already in error while the subroutine is running, the subroutine is called again; this will result in a program loop unless the programmer codes the subroutine to avoid this problem. One way to avoid such a program loop is to set a first-time switch in the subroutine. If it is not the first time through the subroutine, set on a halt indicator and issue the RETRN operation as follows:

```
                                                    Comments
-------------------------------------------------------------------------
ERRRTN      BEGSR                                   INFSR
SW          IFEQ        '1'                         ALREADY HANDLING ERROR
            SETON                       H1
            RETRN                                   EXIT
            ELSE
            MOVE        '1'     SW                  FLAG HANDLER ERROR
             .
             .
             .
            END
            MOVE        '0'     SW                  END ERROR PROCESSING
            ENDSR
```

Note: It may not be possible to continue processing the file after an I/O error has occurred. To continue, it may be necessary to issue a CLOSE operation and then an OPEN operation to the file.

Figure 4-9 on page 4-48 shows an example of a file information data structure (INFDS) and a file exception/error subroutine.

File Description Specifications

Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/O/F	E	A/D	F/V/S/M/D/E	Block Length	Record Length	L/R	A/P/I/K	Length of Key Field or Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code	E/L	Device	Symbolic Device	Labels S/N/E/M	Name of Label Exit	Extent Exit for DAM	Storage Index	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8, UC	A/U	R/U/N	K	Option	Entry
02	F	TRNFIL	OF		E						K								DISK													KINFDS	FILDS1
03	F																															KINFSR	ERRRTN
04	F	MSTFIL	UF		E						K								DISK													KINFDS	FILDS2
05	F																															KINFSR	MSTERR
06	F	AUDITFIL	OF		E						K								DISK														

Three files are defined on the file description specifications. You want to control the program logic if an exception/error occurs on the TRNFIL file or on the MSTFIL file. Therefore, a unique file information data structure (INFDS) and a file exception/error subroutine (INFSR) are defined for each file. No file information data structure or file exception/error subroutine is defined for the AUDITFIL file.

Line	Form Type	Filename or Record Name	Sequence	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, ** or DS	External Field Name	Record Identification Codes 1 Position	1 Not (N)	1 C/Z/D	1 Character	2 Position	2 Not (N)	2 C/Z/D	2 Character	3 Position	3 Not (N)	3 C/Z/D	3 Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Data Structure Occurs n Times	Data Structure Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank
01	I	FILDS1				DS																											
02	I																										*FILE	FIL1						
03	I																										*RECORD	REC1						
04	I																										*OPCODE	OP1						
05	I																										*STATUS	STS1						
06	I																										*ROUTINE	RTN1						
07	I	FILDS2				DS																											
08	I																										*FILE	FIL2						
09	I																										*RECORD	REC2						
10	I																										*OPCODE	OP2						
11	I																										*STATUS	STS2						
12	I																										*ROUTINE	RTN2						
13	I																																

The location of the subfields in the file information data structures is defined by special keywords in positions 44 through 51. To access these predefined subfields, a name must be assigned to each subfield in positions 53 through 58. If an exception/error occurs, you can test the information in the data structure to determine, for example, what exception/error occurred (*STATUS) and on which operation it occurred (*OPCODE). You can then use that information within the file exception/error subroutine to determine what action to take.

Figure 4-9 (Part 1 of 2). Example of File Exception/Error Handling

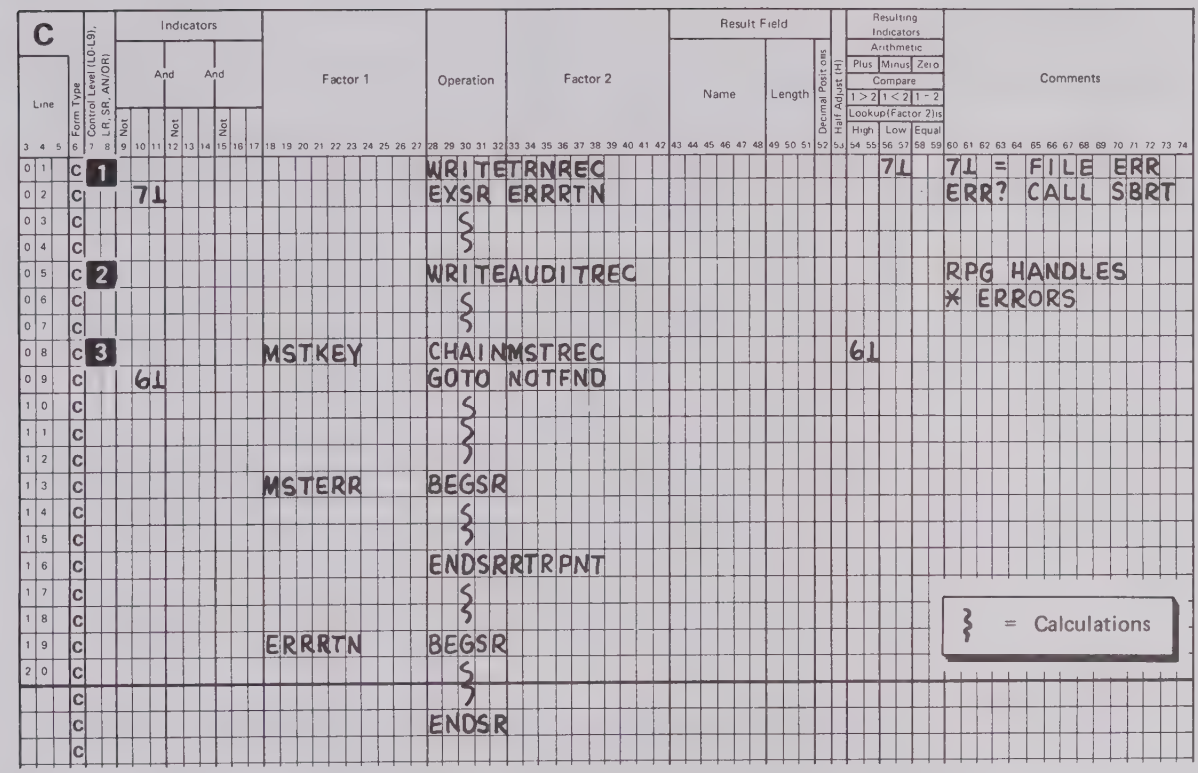


Figure 4-9 (Part 2 of 2). Example of File Exception/Error Handling

1 On the WRITE operation to the TRNREC record in the TRNFIL file, an exception/error indicator is specified in positions 56 and 57. This indicator is set on if an exception/error occurs on this operation. The ERRRTN subroutine (the file exception/error subroutine for the TRNFIL file) is explicitly called by the EXSR operation when indicator 71 is on. Because factor 2 of the ENDSR operation for the ERRRTN is blank, control returns to the next sequential instruction following the EXSR operation after the subroutine has run.

2 No exception/error indicator is specified in positions 56 and 57 of the WRITE operation to the AUDITREC record in the AUDITFIL file. No file exception/error subroutine was defined for this file on the file description specifications. Therefore, any exception/errors that occur on this operation to the AUDITFIL file are handled by the RPG/400 default error handler.

3 No exception/error indicator is specified in positions 56 and 57 for the CHAIN operation to the MSTREC record in the MSTFIL file. However, a file exception/error subroutine (MSTERR) is defined for the file on the file description specifications. Therefore, when an exception/error other than no record found occurs on the CHAIN operation, RPG/400 passes control to the MSTERR subroutine. On the ENDSR operation for this subroutine, factor 2 contains a field name. This allows you to alter the return point from the subroutine within the subroutine based on the exception/error that occurred. The field must contain one of the values described under "File Exception/Error Subroutine (INFSR)" on page 4-45.

Program Exception/Errors

Some examples of program exception/errors are invalid array index, division by zero, SQRT of a negative number, an error on a CALL or FREE operation code, and an error return from a called program. They can be handled in one of the following ways:

- An indicator can be specified in positions 56 and 57 of the calculation specifications for an operation code. This indicator is set on if an exception/error occurs during the processing of the specified file operation. The optional program status data structure is updated with the exception/error information. You can determine the action to be taken by testing the indicator.
- A program exception/error subroutine can be specified. You enter *PSSR in factor 1 of a BEGSR operation to specify this subroutine. Information regarding the program exception/error is made available through a program status data structure that is specified with an S in position 18 of the data structure statement on the input specifications.
- If the indicator or the program exception/error subroutine is not present, program exception/errors are handled by the RPG/400 default error handler.

Program Status Data Structure

You can define a program status data structure to make program exception/error information available to the program.

A data structure is defined as a program status data structure by an S in position 18 of the data structure statement. A program status data structure contains predefined subfields that provides you with information about the program exception/error that occurred.

The location of the subfields in the program status data structure is defined by special keywords or by predefined From and To positions. In order to access the subfields, you assign a name to each subfield in positions 53 through 58.

The keywords must be specified, left-adjusted in positions 44 through 51. The keywords are not labels and cannot be used to access the subfields. Short entries are padded on the right with blanks. The keywords and their descriptions are as follows:

Keyword	Description
*STATUS	Five-digit numeric field, with zero decimal positions that contains the status code. For a description of these codes, see "Status Codes" on page 4-42.

Keyword	Description
*ROUTINE	Eight-position character field that contains the name of the RPG/400 routine in which the exception/error occurred. This subfield is updated at the beginning of an RPG/400 routine or after a program call only when the *STATUS subfield is updated with a nonzero value. The following names identify the routines: *INIT Program initialization *DETL Detail lines *GETIN Get input record *TOTC Total calculations *TOTL Total lines *DETC Detail calculations *OFL Overflow lines *TERM Program ending pgmname Name of program called (first 8 characters). **Note:** *ROUTINE is not valid unless you use the normal RPG/400 cycle. Logic that takes the program out of the normal RPG/400 cycle may cause *ROUTINE to reflect an incorrect value.
*PARMS	Three-digit numeric field that contains the number of parameters passed to this program from the calling program.
*PROGRAM	Ten-position character field that contains the name of the program in which this program status data structure is specified.

Information from the program status data structure is also provided in a formatted dump.

Status Code

Any code placed in the subfield location *STATUS that is greater than 99 is considered to be an exception/error condition. If the status code is greater than 99, the error indicator, if specified in positions 56 and 57, is set on, or the program exception/error subroutine receives control. Location *STATUS is updated when an exception/error occurs.

The following codes are placed in the subfield location *STATUS for the program status data structure:

Normal Codes

Code	Condition
00000	No exception/error occurred.
00001	Called program returned with the LR indicator on.

Exception/Error Codes

Code	Condition
00101	Negative square root
00102	Divide by zero
00121	Invalid array index
00122	OCUR outside of range
00202	Called program failed; halt indicator (H1 through H9) not on
00211	Program specified on CALL or FREE not found
00221	Called program tried to use a parameter that was not passed to it
00231	Called program returned with halt indicator on
00232	Halt indicator on in this program
00233	Halt indicator on when RETRN operation run
00299	RPG/400 formatted dump failed
00333	Error on DSPLY operation
00401	Data area specified on IN/OUT not found
00411	Data area type or length does not match
00412	Data area not locked for output
00413	Error on IN/OUT operation
00414	User not authorized to use data area
00415	User not authorized to change data area
00421	Error on UNLCK operation
00431	Data area previously locked by another program
00432	Data area locked by program in the same process
00907	Decimal data error (invalid digit or sign)
00970	The level number of the compiler used to generate the program does not agree with the level number of the RPG/400 run-time subroutines
09998	Internal failure in RPG/400 compiler or in run-time subroutines
09999	Program exception in OS/400 routine.

Program Exception/Error Subroutine

To identify the user-written RPG/400 subroutine that is to receive control when a program exception/error occurs, specify *PSSR in factor 1 of the subroutine's BEGSR operation. If an indicator is not specified in positions 56 and 57 for the operation code control is transferred to this subroutine when a program exception/error occurs. In addition, the subroutine can also be called by the EXSR operation. *PSSR can be specified in positions 60 through 65 of the file description specifications continuation line for the file exception/error subroutine (INFSR in positions 54 through 59), and receives control if a file exception/error occurs.

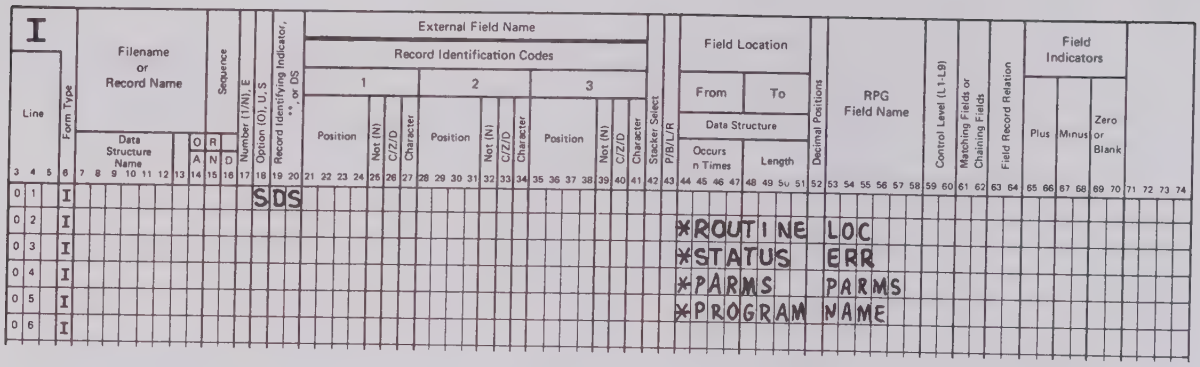
Any of the RPG/400 operation codes can be used in the program exception/error subroutine. The ENDSR operation must be the last specification for the subroutine, and the factor 2 entry on the ENDSR operation specifies the return point following the running of the subroutine. For a discussion of the valid entries for factor 2, see "File Exception/Error Subroutine (INFSR)" on page 4-45.

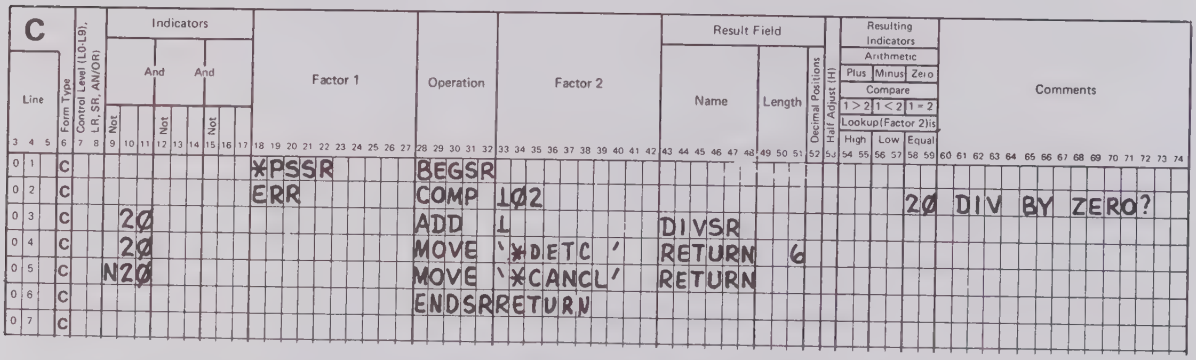
Remember the following when specifying a program exception/error subroutine:

- You can explicitly call the *PSSR subroutine by specifying *PSSR in factor 2 of the EXSR operation.
- After the ENDSR operation of the *PSSR subroutine is run, RPG/400 resets the field or array element specified in factor 2 to blanks. Thus, if you do not place a value in this field during the running of the subroutine, the RPG/400 default error handler receives control following the running of the subroutine, unless the *PSSR subroutine was called by the EXSR operation. This allows you to specify the return point within the subroutine that is best suited for the running/error that occurred. If the subroutine was called by the EXSR operation and factor 2 of the ENDSR operation is blank, control returns to the next sequential instruction following the EXSR operation.
- Because the program exception/error subroutine may receive control whenever a non-file exception/error occurs, an exception/error could occur while the subroutine is running. If an exception/error occurs while the subroutine is running, the subroutine is called again; this will result in a program loop unless the programmer codes the subroutine to avoid this problem. One way to avoid such a program loop is to set a first-time switch in the subroutine. If it is not the first time through the subroutine, set on a halt indicator and issue the RETRN operation as follows:

```
*PSSR       BEGSR
'1'         IFNE        SW
            MOVE        '1'     SW
             .
             .
             .
            ELSE
            SETON                       H1
            RETRN
            END
             .
             .
            MOVE        '0'     SW
            ENDSR
```

Figure 4-10 on page 4-54 shows an example of a program exception/error subroutine.





The program status data structure is defined on the input specifications. The predefined subfields *STATUS, *ROUTINE, *PARMS, and *PROGRAM are specified, and names are assigned to the subfields.

The *PSSR subroutine is coded on the calculation specifications. If a program exception/error occurs, RPG/400 passes control to the *PSSR subroutine. The subroutine checks to determine if the exception/error was caused by a divide operation in which the divisor is zero. If it was, indicator 20 is set on, 1 is added to the divisor (DIVSR), and the literal '*DETC'. is moved to the field RETURN. Moving the literal into the RETURN field, which is specified in factor 2 of the ENDSR operation, allows you to control the return point within the subroutine. In this example, control returns to the beginning of the detail calculations routine unless the exception/error was not a divide by zero. In that case, the literal '*CANCL'. is moved into the RETURN field, and the program is ended.

Figure 4-10. Example of *PSSR Subroutine

Chapter 5. The Detailed RPG Program Cycle

In Chapter 1, "An Introduction to RPG/400 and the AS/400 System" on page 1-1, the basic RPG Logic Cycle was introduced. In Figure 5-1 on page 5-2, Figure 5-2 on page 5-8, Figure 5-3 on page 5-10, and Figure 5-4 on page 5-13, a more detailed explanation of the RPG Logic Cycle is presented.

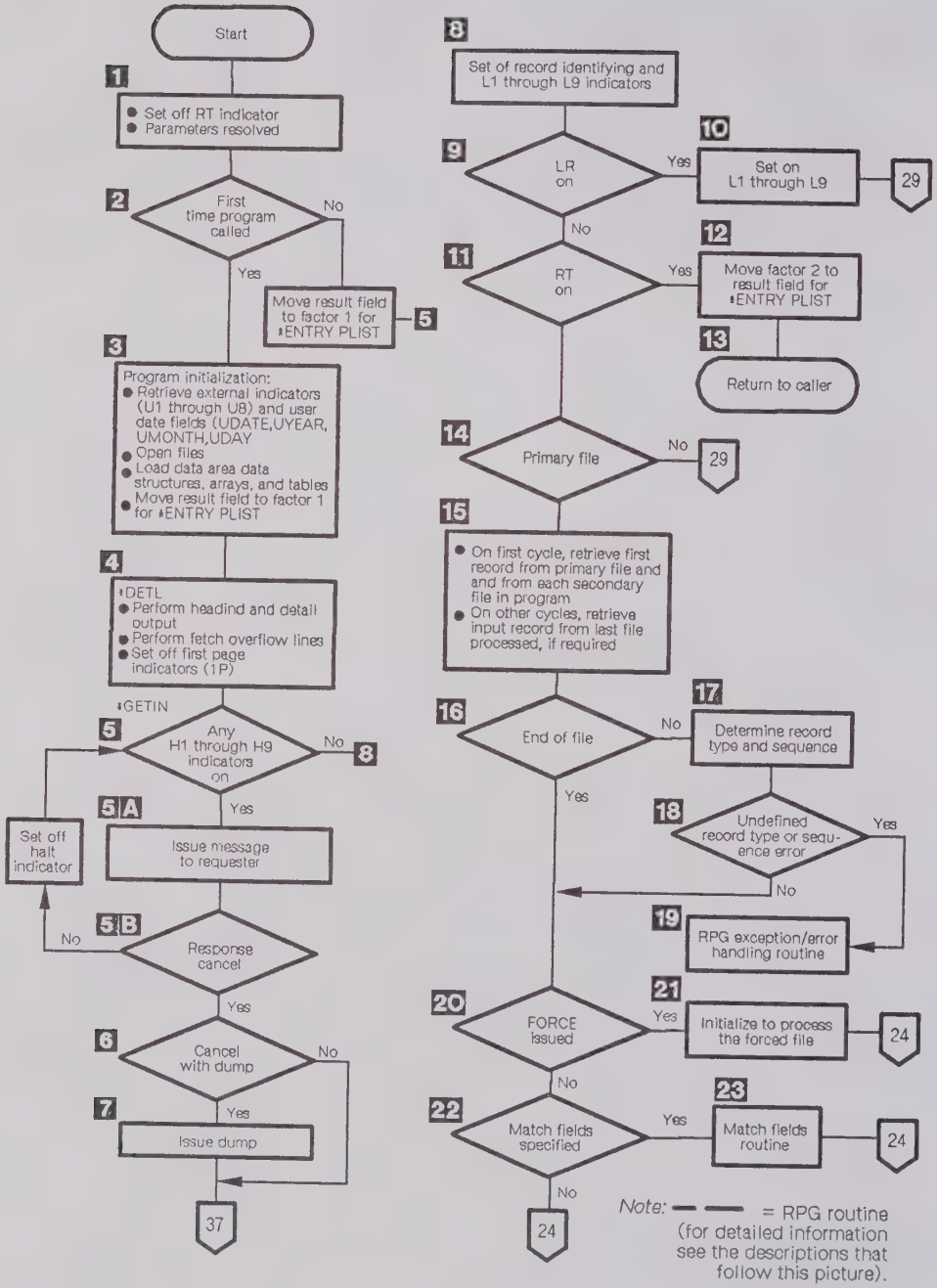


Figure 5-1 (Part 1 of 2). Detailed RPG/400 Object Program Cycle

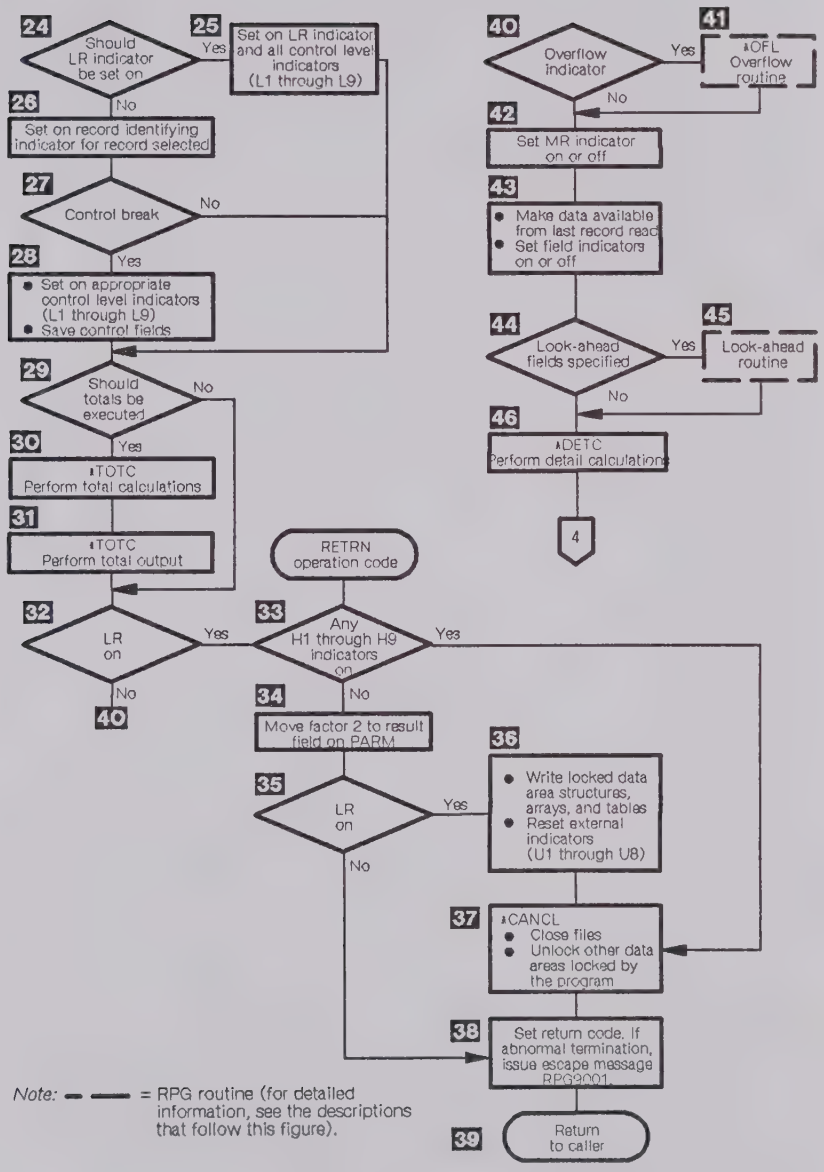


Figure 5-1 (Part 2 of 2). Detailed RPG/400 Object Program Cycle

Detailed RPG/400 Program Cycle

Figure 5-1 on page 5-2 shows the specific steps in the detailed flow of the RPG/400 program cycle. The item numbers in the following description refer to the numbers in the figure. Routines are flowcharted in Figure 5-2 on page 5-8 and in Figure 5-3 on page 5-10.

1 The RT indicator is set off. RPG/400 determines whether *ENTRY PLIST is specified. Parameters are resolved.

2 RPG/400 checks for the first invocation of the program. If it is the first invocation, RPG/400 continues with program initialization. If not, it moves the result field to factor 1 in the PARMS statement in *ENTRY PLIST and branches to step 5.

3 The program is initialized. RPG/400 sets up the external indicators (U1 through U8) and user date fields (UDATE, UYEAR, UMONTH, UDAY), opens the files, loads all data area data structures and arrays and tables, and moves the result field to factor 1 in the PARMS statement in *ENTRY PLIST.

4 Heading and detail lines (identified by an H or D in position 15 of the output specifications) are written before the first record is read. Heading and detail lines are always processed at the same time. If conditioning indicators are specified, the proper indicator setting must be satisfied. If fetch overflow logic is specified and the overflow indicator is on, the appropriate overflow lines are written. File translation, if specified, is done for heading and detail lines and overflow output. This step is the return point in the program if factor 2 of an ENDSR operation contains a field name or a literal with the value *DETL.

5 RPG/400 tests the halt indicators (H1 through H9). If all the halt indicators are off, the program branches to step 8. Halt indicators can be set on anytime during the program. This step is the return point in the program if factor 2 of an ENDSR operation contains a field name or a literal with the value *GETIN.

a. If any halt indicators are on, a message is issued to the requester. For an interactive job, the message goes to the requester. For a batch job, the message goes to QSYSOPR; if QSYSOPR is not in break mode, a default response is issued.

b. If the response is to continue, the halt indicator is set off, and the program returns to step 5. If the response is to cancel, the program goes to step 6.

6 If the response is to cancel with a dump, the program goes to step 7; otherwise, the program branches to step 37.

7 The program issues a dump and branches to step 37 (abnormal ending).

8 RPG/400 sets off all record identifying, 1P (first page), and control level (L1 through L9) indicators. RPG/400 sets all overflow indicators (OA through OG, OV) off unless they have been set on during preceding detail calculations or detail output. Any other indicators that are on remain on.

9 RPG/400 determines whether the LR (last record) indicator is on. If it is on, the program continues with step 10. If it is not on, the program branches to step 11.

10 RPG/400 sets the appropriate control level (L1 through L9) indicators on and branches to step 29.

11 RPG/400 determines whether the RT indicator is on. If it is on, the program continues with step 12; otherwise, the program branches to step 14.

12 RPG/400 moves factor 2 to the result field for *ENTRY PLIST.

13 If the RT indicator is on, the called program returns to the caller.

14 RPG/400 determines whether the program contains a primary file. If a primary file is present in the program, the program continues with step 15; otherwise, the program branches to step 29.

15 During the first program cycle, RPG/400 reads the first record from the primary file and from each secondary file in the program. In other program cycles, RPG/400 reads a record from the last file processed. If this file is processed by a record address (RA) file, the data in the record address file defines the record to be retrieved. If look-ahead fields are specified in the last record processed, the record may already be in storage; therefore, no read may be done at this time.

16 RPG/400 determines whether end of file has occurred on the file just read. If it has not occurred, the program continues with step 17; otherwise, the program branches to step 20.

17 If RPG/400 has read a record from the file, the record type and record sequence (positions 15 through 18 of the input specifications) are determined.

18 RPG/400 determines whether the record type is defined in the program, or if the record sequence is correct. If the record type is undefined or the record sequence is incorrect, the program continues with step 19; otherwise, the program branches to step 20.

19 The RPG/400 exception/error handling routine receives control. (For detailed information on the RPG/400 exception/error handling routine, see "RPG Exception/Error Handling Routine" on page 5-8 and see "Exception/Error Handling" on page 4-35.)

20 RPG/400 determines whether a FORCE operation was processed on the previous cycle. If a FORCE operation was processed, the program selects that file for processing (step 21) and branches around the processing for match fields (steps 22 and 23). The branch is processed because all records processed with a FORCE operation are processed with the matching record (MR) indicator off.

21 If FORCE was issued on the previous cycle, the program selects the forced file for processing after removing any match fields from the file just read. If the file forced is at end of file, normal primary/secondary multifile logic selects the next record for processing and the program branches to step 24.

22 RPG/400 determines whether match fields are specified. If they are, the program continues with step 23; otherwise, the program branches to step 24.

23 The match fields routine receives control. (For detailed information on the match fields routine, see "Match Fields Routine" on page 5-10.)

24 RPG/400 sets on the LR (last record) indicator when all records are processed from the files that have an E specified in position 17 of the file description specifications and all matching secondary records have been processed. If the LR indicator is not set on, processing continues with step 26.

25 RPG/400 sets on the LR (last record) indicator and all control level (L1 through L9) indicators, and processing continues with step 29.

26 RPG/400 sets on the record identifying indicator for the record selected for processing.

27 RPG/400 determines whether the record selected for processing caused a control break. A control break occurs when the value in the control fields of the record being processed differs from the value of the control fields of the last record processed. If a control break has not occurred, the program branches to step 29.

28 When a control break occurs, RPG/400 sets the appropriate control level indicator (L1 through L9) on. This causes all lower level control indicators to be set on. The program saves the contents of the control field.

29 RPG/400 determines whether the total-time calculations and total-time output should be done. Totals are bypassed on the first cycle if control levels are not specified in the input specifications. After the first cycle, totals are processed on every cycle. If control levels are specified in the input specifications, totals are bypassed until after the first record containing control fields has been processed. Totals are always processed when the LR indicator is on.

30 RPG/400 processes all total calculations conditioned by a control level entry (positions 7 and 8 of the calculation specifications). This step is the return point in the program if factor 2 of an ENDSR operation contains a field name or a literal with the value *TOTC.

31 RPG/400 processes all total output. If fetch overflow logic is specified and the overflow indicator (OA through OG, OV) associated with the file is on, the overflow lines are written. File translation, if specified, is done for all total output and overflow lines. This step is the return point in the program if factor 2 of an ENDSR operation contains a field name or a literal with the value *TOTL.

32 If LR is on, the program continues with step 33; otherwise, the program branches to step 40.

33 RPG/400 tests the halt indicators (H1 through H9). If any halt indicators are on, the program branches to step 37 (abnormal ending). If the halt indicators are off, the program continues with step 34. If the RETRN operation code is used in calculations, the program branches to step 33 upon processing of that operation.

34 RPG/400 moves the factor 2 fields to the result fields on the PARMs.

35 If LR is on, the program continues with step 36. If it is not on, the program branches to step 38.

36 RPG/400 writes all arrays or tables for which a file name is specified in the extension specifications (positions 19 through 26) and writes all locked data area data structures. RPG/400 also resets external indicators (U1 through U8). Output arrays and tables are translated, if necessary.

37 RPG/400 closes all open files. If factor 2 of an ENDSR operation contains a field name or a literal with the value *CANCL, the return point in the program includes steps 37 through 39. RPG/400 also unlocks all data areas that have been locked (*NAMVAR DEFN statement) but not unlocked by the program.

38 The internal return code is set. RPG/400 issues escape message RPG9001 if ending is abnormal.

39 RPG/400 returns control to the caller.

Note: Steps 32 through 39 constitute the normal ending routine. For an abnormal ending, steps 34 through 36 are bypassed.

40 RPG/400 determines whether any overflow indicators (0A through 0G, 0V) are on. If an overflow indicator is on, the program continues with step 41; otherwise, the program branches to step 42.

41 The overflow routine receives control. (For detailed information on the overflow routine, see "Overflow Routine" later in this chapter.) This step is the return point in the program if factor 2 of an ENDSR operation contains a field name or a literal with the value of *OFL.

42 The MR indicator is set on and remains on for the complete cycle that processes the matching record if this is a multifile program and if the record to be processed is a matching record. Otherwise, the MR indicator is set off.

43 Data from the last record read is made available for processing. RPG/400 sets field indicators on, if specified.

44 RPG/400 determines whether look-ahead fields are specified. If so, the program continues with step 45; otherwise, the program branches to step 46.

45 The look-ahead routine receives control. (For detailed information on the look-ahead routine, see "Look-Ahead Routine" on page 5-11.)

46 RPG/400 processes detail calculations. This step is the return point in the program if factor 2 of an ENDSR operation contains a field name or a literal with the value *DETC. The program branches to step 4.

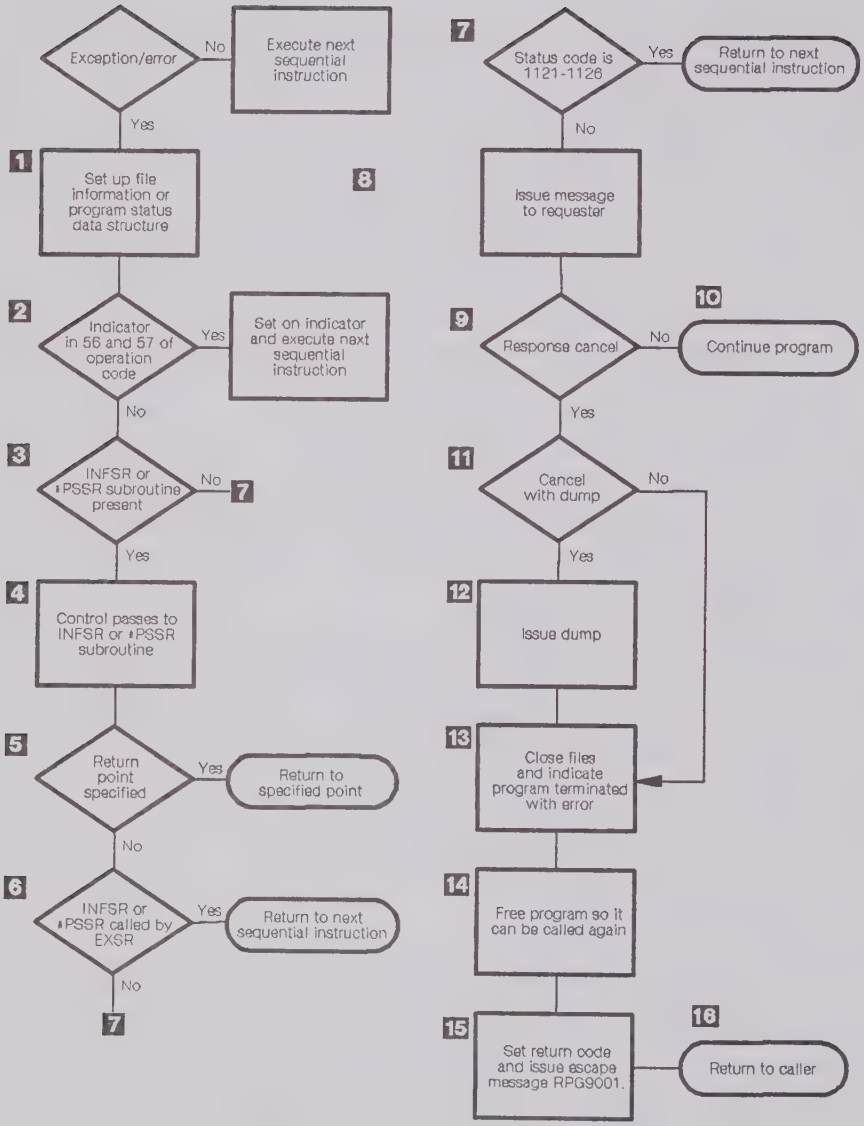


Figure 5-2. Detail Flow of RPG Exception/Error Handling Routine

RPG Exception/Error Handling Routine

Figure 5-2 shows the specific steps in the RPG/400 exception/error handling routine. The item numbers in the following description refer to the numbers in the figure.

1. Set up the file information or program status data structure, if specified, with status information.

2. If the exception/error occurred on an operation code that has an indicator specified in positions 56 and 57, RPG/400 sets the indicator, and control returns to the next sequential instruction in the calculations.

3 RPG/400 determines whether the appropriate exception/error subroutine (INFSR or *PSSR) is present in the program. If it is, the program goes to step 4; if not, the program branches to step 7.

4 Control passes to the exception/error subroutine (INFSR or *PSSR).

5 RPG/400 determines whether a return point is specified in factor 2 of the ENDSR operation for the exception/error subroutine. If a return point is specified, the program goes to the specified return point. If a return point is not specified, the program goes to step 6. If a field name is specified in factor 2 of the ENDSR operation and the content is not one of the RPG/400-defined return points (such as *GETIN or *DETC), the program goes to step 6. No error is indicated, and the original error is handled as though the factor 2 entry were blank.

6 If the exception/error subroutine was called explicitly by the EXSR operation, the program returns to the next sequential instruction. If not, the program continues with step 7.

7 If the Status code is 1121-1126 (see "Status Codes" on page 4-42) control returns to the next sequential instruction in the calculations.

8 A message is issued to the requestor. For an interactive job, the message goes to the requester. For a batch job, the message goes to QSYSOPR; if QSYSOPR is not in break mode, a default response is issued.

9 RPG/400 determines whether the user's response is to cancel the program. If the response is to cancel, the program branches to step 11. If not, the program continues with step 10.

10 The program continues processing at *GETIN.

11 RPG/400 determines whether the user's response is to cancel with a dump. If the response is to cancel with a dump, the program continues with step 12. If not, the program branches to step 13.

12 RPG/400 issues a dump.

13 RPG/400 closes all files and sets the return code to indicate that the program ended with an error.

14 RPG/400 frees the program so that it can be called again, and the program returns to the caller.

15 Set return code and issue escape message RPG9001.

16 Return to caller.

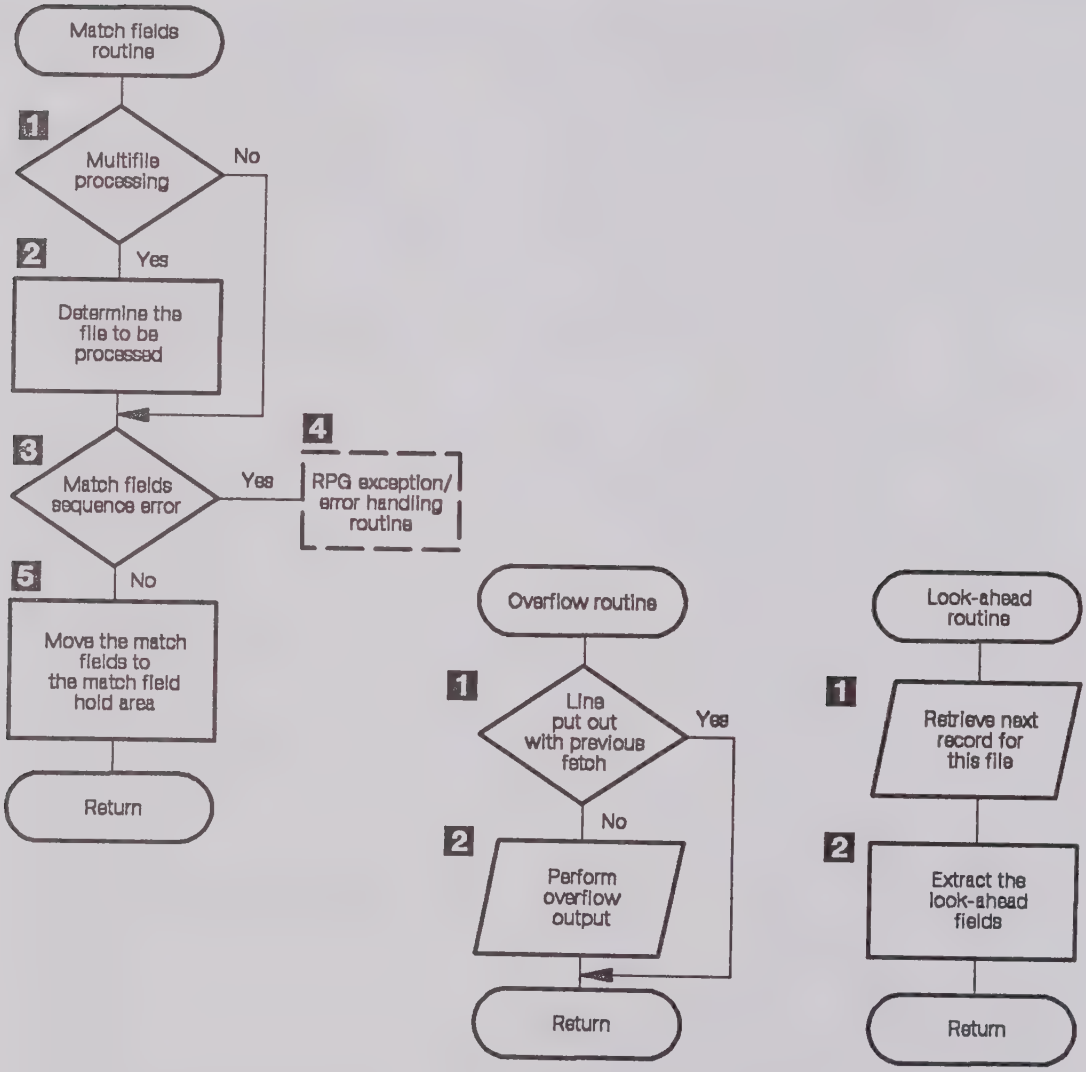


Figure 5-3. Detail Flow of RPG/400 Match Fields, Overflow, and Look-Ahead Routines

Match Fields Routine

Figure 5-3 shows the specific steps in the RPG/400 match fields routine. The item numbers in the following descriptions refer to the numbers in the figure.

1. RPG/400 determines whether multifile processing is being used. If multifile processing is being used, processing continues with step 2; otherwise, the program branches to step 3.
2. RPG/400 tests the value of the match fields in the hold area to determine which file is to be processed next.
3. RPG/400 extracts the match fields from the match files and processes sequence checking. If the match fields are in sequence, the program branches to step 5.

4 If the match fields are not in sequence, the RPG/400 exception/error handling routine receives control.

5 RPG/400 moves the match fields to the hold area for that file. A hold area is provided for each file that has match fields. The next record is selected for processing based on the value in the match fields.

Overflow Routine

Figure 5-3 on page 5-10 shows the specific steps in the RPG/400 overflow routine. The item numbers in the following descriptions refer to the numbers in the figure.

1 RPG/400 determines whether the overflow lines were written previously using the fetch overflow logic (step 30 in Figure 5-1 on page 5-2). If the overflow lines were written previously, the program branches to the specified return point; otherwise, processing continues with step 2.

2 RPG/400 tests all output lines conditioned with an overflow indicator and writes the conditioned overflow lines.

Look-Ahead Routine

Figure 5-3 shows the specific steps in the RPG/400 look-ahead routine. The item numbers in the following descriptions refer to the numbers in the figure.

1 RPG/400 reads the next record for the file being processed. However, if the file is a combined or update file (identified by a C or U, respectively, in position 15 of the file description specifications), RPG/400 extracts the look-ahead fields from the current record being processed.

2 RPG/400 extracts the look-ahead fields.

Ending a Program without a Primary File

If your program does not contain a primary file, you *must* specify that the program returns control to its calling program in one of the following ways:

- By setting the LR indicator on
- By setting the RT indicator on
- By setting an H1 through H9 indicator on
- By specifying the RETRN operation code
- By allowing an exception/error to end the program if:
  - No exception/error subroutine is specified
  - No return point is specified on an exception/error subroutine
  - The user's response is to cancel the program.

For more information on the RPG/400 exception/error handling routine, see "Exception/Error Handling" on page 4-35.

See Chapter 11, "Communicating with Objects in the System" for a description of how the LR, RT, H1 through H9 indicators, the RETRN operation code, and the

exception/error routine can be used in conjunction with each other, and for a detailed description of the ways to return control to the calling program.

Program Control of File Processing

Specify a full procedural file (F in position 16 of the file description specifications) to control all or partial input of a program. A full procedural file indicates to the compiler that *input* is controlled by program-specified calculation operations (for example, `READ`, `CHAIN`). When both full procedural files and a primary file (P in position 16 of the file description specifications) are specified in a program, some of the input is controlled by the program, and other input is controlled by the cycle. The program cycle exists when a full procedural file is specified; however, file processing occurs at detail or total calculation time.

The file operation codes can be used for program control of input. These file operation codes are discussed in *Languages: RPG/400 Reference.*

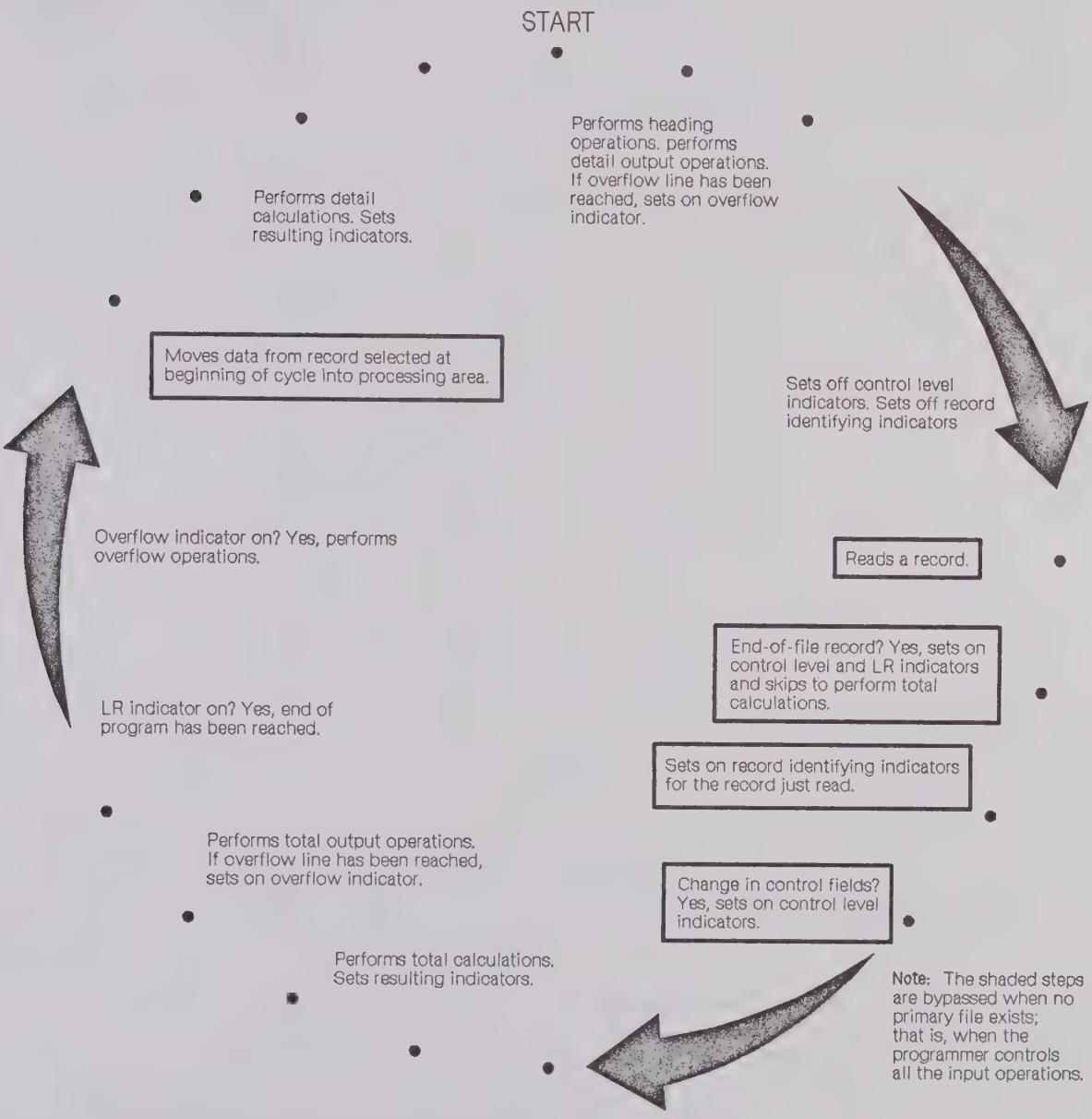


Figure 5-4. Programmer Control of Input Operation within the Program-Cycle

For an example of program control of input, see the WORKSTN file examples in Chapter 9, "Using WORKSTN files."

Chapter 6. General File Considerations

This chapter describes:

- The device independent and device dependent characteristics of RPG/400 on the AS/400 System
- AS/400 spooling functions
- The extent to which externally described and program described files are defined in RPG/400
- Level checking functions
- File locking by RPG/400
- Record locking by RPG/400
- Unblocking/blocking records to improve performance
- Sharing an open data path
- General information about the use of externally described files and how this external description can be modified in the RPG/400 program
- Program described files
- RPG/400 functions that relate specifically to an RPG/400 PRINTER device, SEQ device, and SPECIAL device.
- Primary/secondary multifile processing.

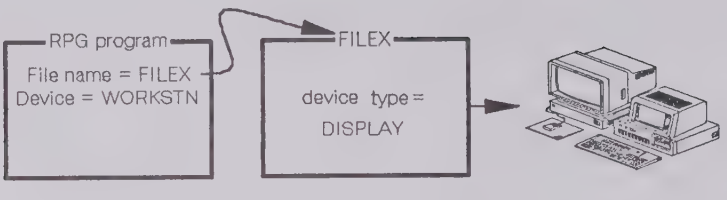
On the AS/400 System files are made up of members. These files are organized into libraries. The convention for naming files is `library-name/file-name`.

Device Independence/Device Dependence

The key element for all input/output operations is the file. All files used on the system are defined to OS/400. OS/400 maintains a description of each file that is accessed by a program when it uses the file.

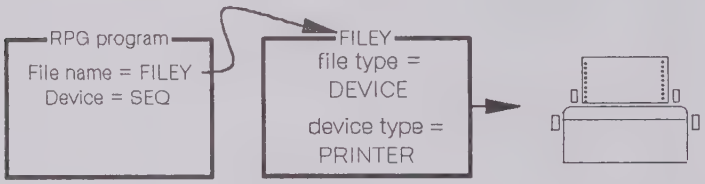
The OS/400 file descriptions are kept online and serve as the connecting link between a program and the device used for I/O. The data is read from or written to the device when the file is used for processing. In some instances, this type of I/O control allows you to change the type of file (and, in some cases, change the device) used in a program without changing the program.

On the AS/400 System, the file name specified in positions 7 through 14 of the file description specification is used to point to the file, rather than the device name specified in positions 40 through 46. The file name points to the OS/400 file description that contains the specifications for the actual device:

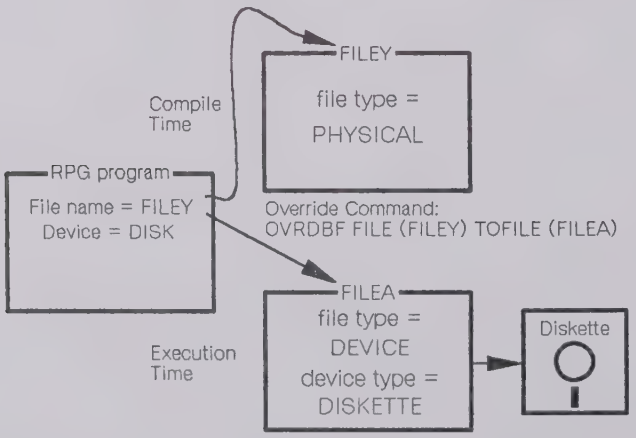


The RPG/400 device name in positions 40 through 46 defines the RPG/400 functions that can be processed on the associated file. At compilation time, certain RPG/400 functions are valid only for a specific RPG/400 device name; therefore, in this respect, RPG/400 is device dependent. One example of device dependency is that the EXFMT operation code is valid only for a WORKSTN device.

For another example, assume that the file name FILEY is specified in the RPG/400 program with the SEQ device. The device SEQ is an independent device type. When the program is run, the actual I/O device is specified in the description of FILE; for example, the device might be PRINTER.



OS/400 commands can be used to override a parameter in the specified file description or to redirect a file at compilation time or run time. File redirection allows you to specify one file at compilation time and another file at run time:



In the preceding example, the CL command OVRDBF (Override With Data Base File) allows the program to run with an entirely different device file than was specified at compilation time.

However, not all file redirections or overrides are valid. At run time, checking occurs to ensure that the specifications within the RPG/400 program are valid for the

file being processed. OS/400 allows some file redirections even if device specifics are contained in the program. For example, if the RPG/400 device name is PRINTER and the actual file the program connects to is not a printer, OS/400 ignores the RPG/400 print spacing and skipping specifications. However, there are other file redirections that OS/400 does not allow and that cause the program to end. For example, if the RPG/400 device name is WORKSTN and the EXFMT operation is specified in the program, the program is stopped if the actual file the program connects to is not a display or ICF file.

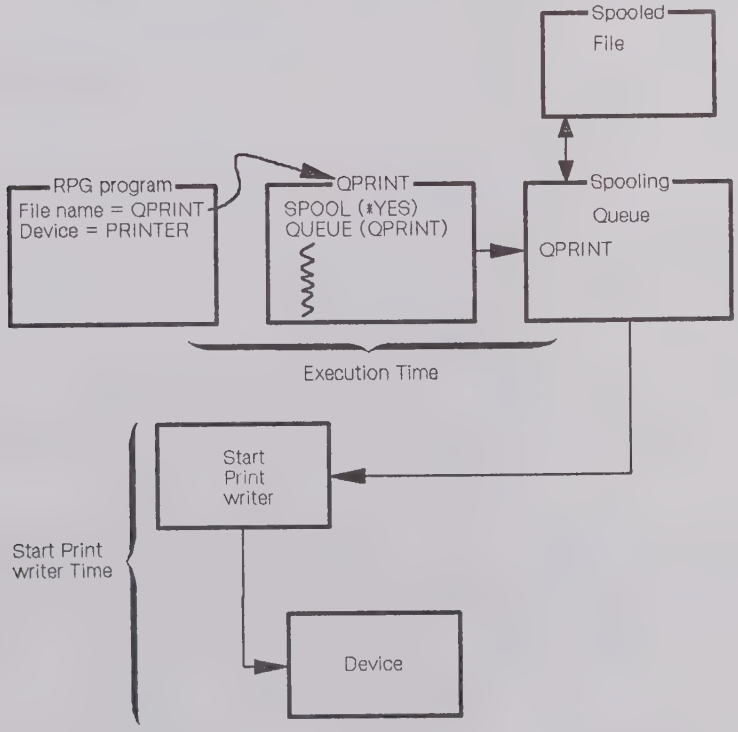
See the *Programming: Data Management Guide* for more detailed information on valid file redirections and file overrides.

Spooling

Spooling is a system function that puts data into a storage area to wait to be processed. The AS/400 System provides for the use of input and output spooling functions. The RPG/400 program is not aware that spooling is being used. The actual physical device from which a file is read or to which a file is written is determined by the spool reader or the spool writer. For more detailed information on spooling, see the *Programming: Data Management Guide.*

Output Spool

Output spooling is valid for batch or interactive jobs. The description of the file that is specified in RPG/400 by the file name contains the specification for spooling as shown in the following example:



File override commands can be used at run time to override the spooling options that are specified in the file description, such as the number of copies to be printed. In addition, AS/400 spooling support allows a user to redirect a file after the program has run. The user can direct the same printed output to a different device such as a diskette.

Externally Described/Program Described Files

All files on the AS/400 System are defined to OS/400. However, the extent to which files can be defined differs:

- An *externally described file* is described to the OS/400 at the field level. The description includes information about where the data comes from, such as the data base or a specific device, and a description of each field and its attributes.
- A *program described file* is described at the field level within the RPG/400 program on input/output specifications. The description of the file to the OS/400 includes information about where the data comes from and the length of the records in the file.

An externally described file does not have to be redefined in an RPG/400 program on input/output specifications. However, the fields, and their attributes, in a program described file must be described on input/output specifications.

Externally described files offer the following advantages:

- Less coding in RPG/400 programs. If the same file is used by many programs, the fields can be defined once to the OS/400 and used by all the programs. This eliminates the need to code input and output specifications for RPG/400 programs that use externally described files.
- Less maintenance activity when the file's record format is changed. You can often update programs by changing the file's record format and then recompiling the programs that use the files without changing any coding in the program.
- Improved documentation because programs using the same files use consistent record format and field names.

If an externally described file (identified by an E in position 19) is specified for the devices SEQ, or SPECIAL, RPG/400 uses the field descriptions for the file, but the interface to OS/400 is as though the file were a program described file. Externally described files cannot specify device-dependent functions such as forms control.

You can choose, to use an externally described file within the program by specifying the file as program described (F in position 19 of the file description specifications). In this case, the compiler does not copy in the external field-level description of the file at compilation time. This approach is used in conversion where existing programs use program described files and new programs use externally described files to refer to the same file.

Figure 6-1 on page 6-5 shows some typical relationships between an RPG/400 program and files on the AS/400 System.

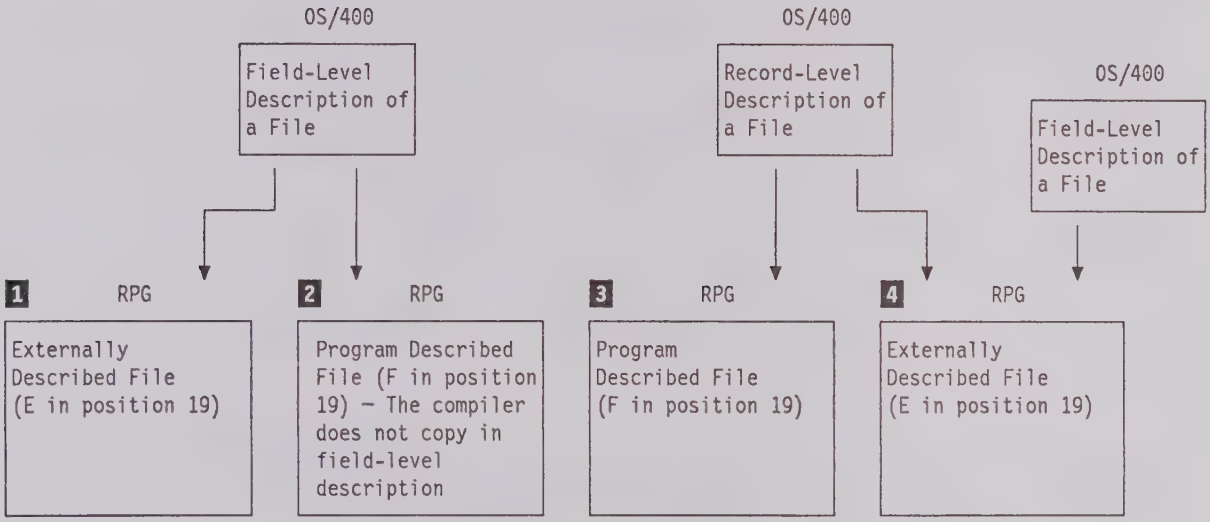


Figure 6-1. Typical Relationships between RPG/400 and Files on the AS/400 System

1 The RPG/400 program uses the field-level description of a file that is defined to the OS/400. An externally described file is identified by an E in position 19 of the file description specifications. At compilation time, the compiler copies in the external field-level description.

2 An externally described file is used as a program described file in the RPG/400 program. A program described file is identified by an F in position 19 of the file description specifications. This entry tells the compiler not to copy in the external field-level descriptions. This file does not have to exist at compilation time.

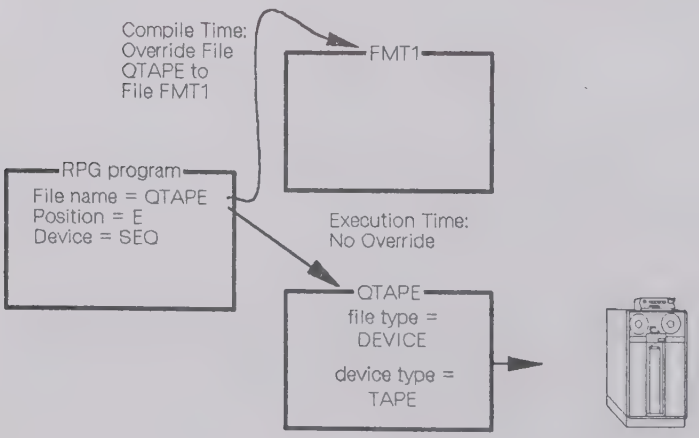
3 A file is described only to the record level to OS/400. The fields in the record are described within the RPG/400 program; therefore, position 19 of the file description specifications must contain an F. This file does not have to exist at compilation time.

4 A file name can be specified for compilation time, and a different file name can be specified for run time. The E in position 19 of the file description specifications indicates that the external description of the file is to be copied in at compilation time. At run time a file override command can be used so that a different file is accessed by the program. However, to override a file at run time, the record names in both files must be the same, because RPG/400 uses the record format name on the input/output operations: such as a READ operation where it specifies what record type is expected.

The following example shows the use of a file override at compilation time. Assume that you want to use an externally described file for a TAPE device, which the system does not support. To do this, you:

- Define a physical file named FMT1 with one record format that contains the description of each field in the record format. The record format is defined on data description specifications (DDS). For a tape device the externally described file should contain only one record format.
- Create the file named FMT1 with a Create Physical File CL command.
- Specify the file name of QTAPE (which is the IBM-supplied device file name for magnetic tape devices) in the RPG/400 program, this identifies the file as externally described (indicated by an E in positions 19 of the file description specifications), and specifies the device name SEQ in positions 40 through 46.
- Use an override command—OVRDBF FILE(QTAPE) TOFILE(FMT1)—at compilation time to override the QTAPE file name and use the FMT1 file name. This causes the compiler to copy in the external description of the FMT1 file, which describes the record format to the RPG/400 compiler.
- Create the RPG/400 program using the CRTRPGPGM command.
- Call the program at run time. The override to file FMT1 should not be in effect while the program is running. Use the CL command DLTOVR (Delete Override).

  Note: You may need to use the CL command OVRTAPF prior to calling the program to provide information necessary for opening the tape file.



Level Checking

Because RPG/400 programs are dependent upon receiving an externally described file whose format agrees with what was copied into the program at compilation time, the system provides a level check function that ensures that the format is still the same.

RPG/400 always provides the information required by level checking when an externally described DISK, WORKSTN, or PRINTER file is used. The level check function can be requested on the create, change, and override file commands. The default on the create file command is to request level checking. Level checking occurs on a record format basis when the file is opened unless you specify LVLCHK(*NO) when you issue an override command or create a file. If the level check values do not match, the program is notified of the error. RPG/400 then handles the OPEN error as described in "Exception/Error Handling" on page 4-35.

RPG/400 does not provide level checking for program described files or for files using the devices SEQ or SPECIAL.

For more information on how to specify level checking, see the *Data Management Guide*.

File Locking by RPG/400

OS/400 allows a lock state (exclusive, exclusive allow read, shared for update, shared no update, or shared for read) to be placed on a file used during a job. Programs within a job are not affected by file lock states. A file lock state applies only when a program in another job tries to use the file concurrently. The file lock state can be allocated with the CL command ALCOBJ (Allocate Object). For more information on allocating resources, and lock states, see the *Programming: Data Management Guide*.

OS/400 places the following lock states on data base files when it opens the files:

File Type	Lock State
Input	Shared for read
Update	Shared for update
Add	Shared for update
Output	Shared for update

The shared-for-read lock state allows another user to open the file with a lock state of shared for read, shared for update, shared no update, or exclusive allow read, but the user cannot specify the exclusive use of the file. The shared-for-update lock state allows another user to open the file with shared-for-read or shared-for-update lock state.

RPG/400 places an exclusive-allow-read lock state on device files. Another user can open the file with a shared-for-read lock state.

The lock state placed on the file by RPG/400 can be changed if you use the Allocate Object command.

Record Locking by RPG/400

When a record is read by RPG/400 for update, a lock is placed on that record so that it is not available for update by any other program, although another program can read the record if the file is coded for input only. Updating that record or reading another record from the file releases that record from its locked state.

If your program has specified an update file and encounters a record locked through another file in your job or through another job, your read operation will wait until the record is unlocked. If the wait time exceeds the specification of the WAITRCD parameter on the CL command OVRDBF (defaults to 60 seconds), an exception will occur. You can determine that a record lock timeout exception has occurred by examining the File Information Data Structure for the status code 01218 (unable to allocate record).

If no changes are required to that record, you can release it from its locked state by processing output operations defined by output specifications with no field names included. These output operations can be processed by EXCPT output, detail output, or total output.

(There are exceptions to these rules when operating under commitment control.)

Unblocking Input Records and Blocking Output Records

To potentially improve the run time performance of input and output operations, the RPG/400 compiler unblocks input records and blocks output records in SEQ or DISK files if:

- The file is an output-only file (O is specified in position 15 of the file description specifications) and contains only one record format if the file is externally described.
- The file is a combined table file (C is specified in position 15 and T in position 16 of the file description specifications).
- The file is an input-only file (I is specified in position 15 of the file description specifications), contains only one record format if the file is externally described, and uses only the OPEN, CLOSE, FEOD, and READ operation codes.

The RPG/400 compiler generates object program code to block and unblock records for all SEQ or DISK files that satisfy these conditions. However, certain OS/400 restrictions may prevent blocking and unblocking. In those cases, performance is not improved and the input/output feedback area is updated for each input/output operation.

The contents of positions 60 through 65 of the RECNO option in the file description specifications continuation line may be invalid when RPG/400 blocks and unblocks records.

The input/output and device-dependent sections of the file information data structure are not updated after each read or write for files in which the records are blocked and unblocked by RPG/400. The feedback area is updated each time a block of records is transferred. (See "File Information Data Structure" on page 4-23 in Chapter 4, "Error Messages, Testing and Debugging" for more information.)

Sharing an Open Data Path

An open data path is the path through which all input and output operations for a file are defined. Usually a separate open data path is defined each time a file is opened. However, if you specify SHARE(*YES) for the file creation or on an override, the first program's open data path for the file is shared by subsequent programs that open the file concurrently. The position of the current record is kept in the open data path for all programs using the file. Therefore, if you read a record in one program and then read a record in a called program, the record retrieved by the second read depends on whether the open data path is shared. If the open data path is shared, the position of the current record in the called program is determined by the current position in the calling program. If the open data path is not shared, each program has an independent position for the current record.

Sharing an open data path improves performance because OS/400 does not have to create a new open data path. However, sharing an open data path can cause problems. For example, an error is signaled in the following cases:

- If a program sharing an open data path attempts file operations other than those specified by the first open (for example, attempting input operations although the first open specified only output operations)
- If a program sharing an open data path for an externally described file tries to use a record format that the first program ignored
- If a program sharing an open data path for a program described file specifies a record length that exceeds the length established by the first open.

If a program shares the open data path for a primary or secondary file, the program must process the detail calculations for the record being processed before calling another program that shares that open data path. Otherwise, if look-ahead is used or if the call is at total time, sharing the open data path for a primary or secondary file may cause the called program to read data from the wrong record in the file.

For additional information about sharing an open data path, see the *Programming: Data Management Guide*.

Using the CL Command RCLRSC

The Reclaim Resources (RCLRSC) command is designed for use in the controlling program of an application. It frees the static storage and closes any files that were left open by other programs in the application that are no longer active. This command will not always free program static storage or close all files. Using RCLRSC may close some files but retain their static storage. When this occurs, static storage indicates that these files are open, but their open data path (ODP) does not exist.

When I/O is attempted with these files an error will occur. For additional information refer to the *Programming: Control Language Reference.*

RPG Specifications for Externally Described Files

You can use data description specifications (DDS) to describe files to OS/400. Each record type in an externally described file is identified by a unique record format name. Figure 6-2 on page 6-11 shows an example of the DDS for a file with one record format (one record type).

The following text describes the special entries that you can use on the file description, input, and output specifications for externally described files. Remember that input and output specifications for externally described files are optional.

File Description Specifications

An E entry in position 19 of the file description specifications identifies an externally described file. The E entry indicates to the compiler that it is to retrieve the external description of the file from the system when the program is compiled.

The information in this external description includes:

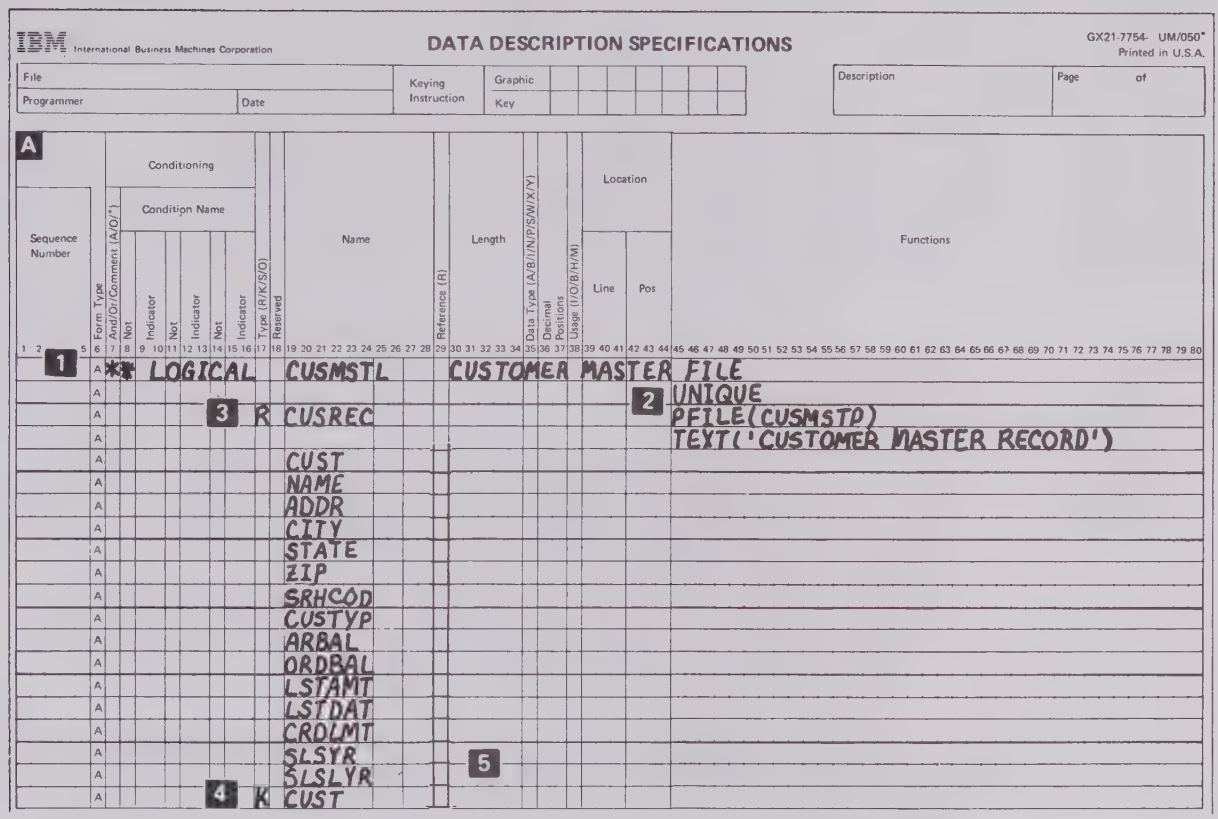
- File information, such as file type, and file attributes, such as access method (by key or relative record number)
- Record format description, which includes the record format name and field descriptions (names, locations, and attributes).

The information RPG/400 retrieves from the external description is printed on the compiler listing when the program is compiled.

Renaming Record Format Names

Many of the functions that you can specify for externally described files (such as the CHAIN operation) operate on either a file name or a record format name. Consequently, each file and record format name in the program must be a unique symbolic name.

To rename a record format name, use the RENAME option on the file description specifications continuation line for the externally described file (see Figure 6-3 on page 6-12). You cannot specify the RENAME option on the main file description specifications line. The RENAME option is generally used if the program contains two identical record format names or if the record format name exceeds 8 characters, which is the maximum length allowed in RPG/400.



IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754- UM/050* Printed in U.S.A.

File | Keying Instruction | Graphic | Description | Page of

Programmer | Date | Key

A

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not	Indicator	Not	Indicator	Not	Indicator	Type (R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (A/B/I/N/P/S/W/X/Y)	Decimal Positions	Usage (I/O/B/H/M)	Location: Line	Location: Pos	Functions
1 2 5	6	7	8	9 10	11 12	13 14	15 16		17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
1	A	**									LOGICAL		CUSMSTL						CUSTOMER MASTER FILE
	A																	2	UNIQUE
	A					3			R		CUSREC								PFILE(CUSMSTP)
	A																		TEXT('CUSTOMER MASTER RECORD')
	A										CUST								
	A										NAME								
	A										ADDR								
	A										CITY								
	A										STATE								
	A										ZIP								
	A										SRHCOD								
	A										CUSTYP								
	A										ARBAL								
	A										ORDBAL								
	A										LSTAMT								
	A										LSTDAT								
	A										CRDLMT								
	A										SLSYR		5						
	A										SLSLYR								
	A					4			K		CUST								

Figure 6-2. Example of Data Description Specifications

1. A logical file for processing the customer master physical file (CUSMSTP) is defined and named CUSMSTL.
2. The UNIQUE keyword indicates that duplicate key values for this file are not allowed.
3. One record format (CUSREC) is defined for the CUSMSTL file.
4. The CUST field is identified as the key field for this file.
5. If field attributes (such as length, data type, and decimal positions) are not specified in the DDS for a logical file, the attributes are obtained from the corresponding field in the physical file. Any field attributes specified in the DDS for the logical file override the attributes for the corresponding field in the physical file. The definition of the fields in the physical file could refer to a field reference file. A field reference file is a data description file consisting of field names and their definitions, such as size and type. When a field reference file is used, the same fields that are used in multiple record formats have to be defined only once in the field reference file. For more information on a field reference file, see the *Programming: Data Base Guide*.

File Description Specifications

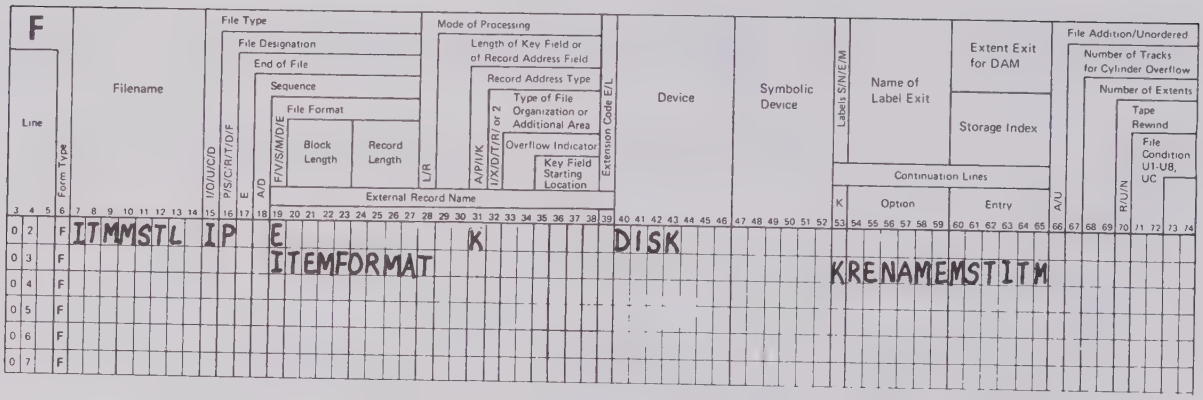


Figure 6-3. RENAME Option for Record Format Names in an Externally Described File

To rename a record format in an externally described file, use a file description specifications continuation line to specify the RENAME option. (The RENAME option cannot be specified on the main file description line because the external name positions overlap some of the entries on the main file description line.) On the continuation line, enter the external name of the record format, left-adjusted, in positions 19 through 28. Specify K in position 53, RENAME in positions 54 through 59, and the program name for the record format, left-adjusted, in positions 60 through 67. The remaining positions of the continuation line must be blank.

In this example, the record format ITEMFORMAT in the externally described file ITMMSTL is renamed MSTITM for use in this program.

Ignoring Record Formats

If a record format in an externally described file is not to be used in a program, you can use the IGNORE option to make the program run as if the record format did not exist in the file. For logical files, this means that all data associated with that format is inaccessible to the program. Use the IGNORE option on a file description specifications continuation line for the externally described file (see Figure 6-4 on page 6-13).

The file must have more than one record format, and not all of them can be ignored; at least one must remain. Except for that requirement, any number of record formats can be ignored for a file.

If a record format name is specified on a continuation line for the IGNORE option, it cannot be specified on a continuation line for any other option (SFILE, RENAME, or PLIST) or on a continuation line for another IGNORE.

Ignored record format names appear on the cross-reference listing, but they are flagged as ignored.

File Description Specifications

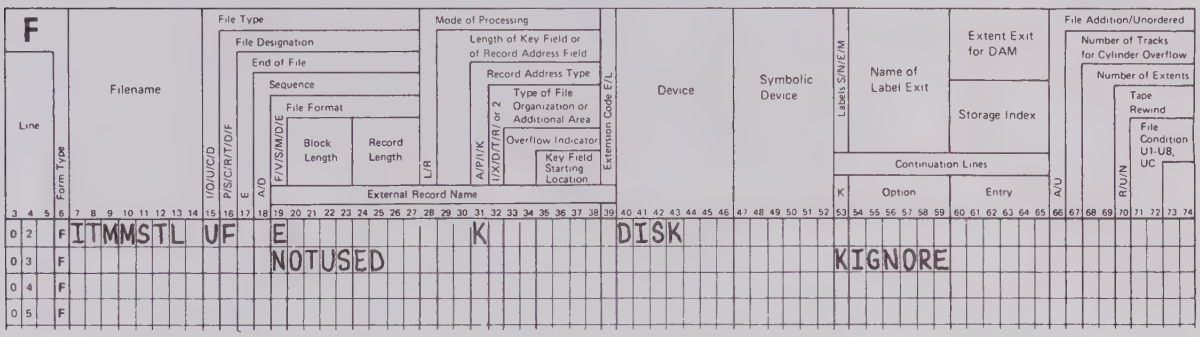


Figure 6-4. IGNORE Option for Record Formats in an Externally Described File

To ignore a record format from an externally described file, use a file description specifications continuation line to specify the IGNORE option. (The IGNORE option cannot be specified on the main file description line because the external name positions overlap some of the entries on the main file description line.) On the continuation line, enter the external name of the record format, left-adjusted, in positions 19 through 28, K in position 53, and IGNORE in positions 54 through 59. The remaining positions of the continuation line must be blank.

In this example, the record format NOTUSED in the externally described file ITMMSTL is ignored.

Floating-Point Fields

RPG/400 does not support the use of floating-point fields. If you process an externally described file with floating-point fields, the following happens:

- You cannot access the floating-point fields.
- When you create a new record, the floating-point fields in the record have the value zero.
- When you update existing records, the floating-point fields are unchanged.
- If you attempt to use a floating-point field as a key field, your program receives a compile-time error.

Overriding or Adding RPG/400 Functions to the External Description

You can use the input specifications to override certain information in the external description of an input file or to add RPG/400 functions to the external description. On the input specifications, you can:

- Assign record identifying indicators to record formats (see Figure 6-5 on page 6-14)
- Rename a field (see Figure 6-5 on page 6-14)
- Assign control level indicators to fields (see Figure 6-5 on page 6-14)
- Assign match field values to fields for matching record processing (see Figure 6-6 on page 6-15)
- Assign field indicators (see Figure 6-6 on page 6-15).

You cannot use the input specifications to override field locations in an externally described file. The fields in an externally described file are placed in the records in the order in which they are listed in the data description specifications. Also, device-dependent functions such as forms control are not valid in RPG/400 for externally described files.

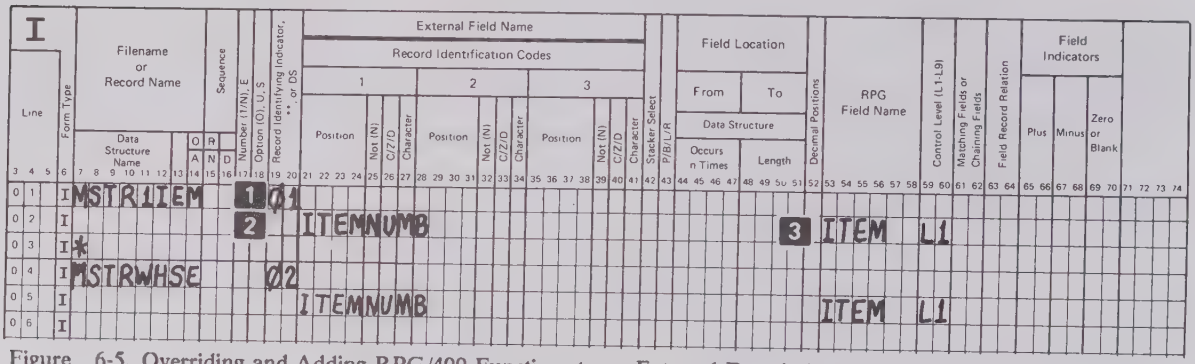


Figure 6-5. Overriding and Adding RPG/400 Functions to an External Description

1 To assign a record identifying indicator to a record in an externally described file, specify the record format name in positions 7 through 14 of the input specifications and assign a valid record identifying indicator in positions 19 and 20. A typical use of input specifications with externally described files is to assign record identifying indicators.

In this example, record identifying indicator 01 is assigned to the record MSTRITEM and indicator 02 to the record MSTRWHSE.

2 To rename a field in an externally described record, specify the external name of the field, left-adjusted, in positions 21 through 30 of the field description line. In positions 53 through 58, specify the name that is to be used in the program.

In this example, the field ITEMNUMB in both records is renamed ITEM for this program because ITEMNUMB exceeds the maximum length of 6 characters that is allowed for an RPG/400 field name.

3 To assign a control level indicator to a field in an externally described record, specify the name of the field in positions 53 through 58 and specify a control level indicator in positions 59 and 60.

In this example, the ITEM field in both records MSTRITEM and MSTRWHSE is specified to be the L1 control field.

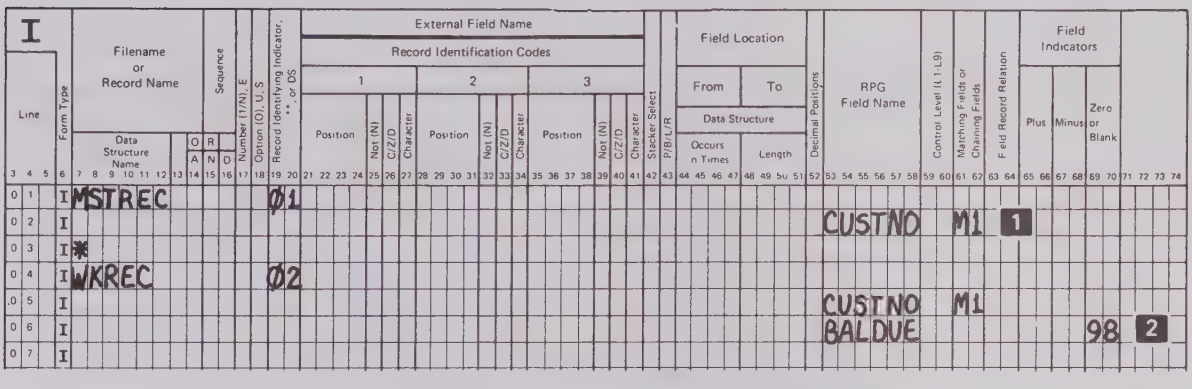


Figure 6-6. Adding RPG/400 Functions to an External Description

1 To assign a match value to a field in an externally described record, specify the record format name in positions 7 through 14 of the record identification line. On the field description line specify the name of the field in positions 53 through 58 and assign a match level value in positions 61 and 62.

In this example, the CUSTNO field in both records MSTREC and WKREC is assigned the match level value M1.

2 To assign a field indicator to a field in an externally described record, specify the record format name in positions 7 through 14 of the record identification line. On the field description line, specify the field name in positions 53 through 58, and specify an indicator in positions 65 through 70.

In this example, the field BALDUE in the record WKREC is tested for zero when it is read into the program. If the field's value is zero, indicator 98 is set on.

Output Specifications

Output specifications are optional for an externally described file. RPG/400 supports file operation codes such as WRITE and UPDAT that use the external record format description to describe the output record without requiring output specifications for the externally described file.

You can use output specification to control when the data is to be written, or to selectively specify the fields that are to be written. The valid entries for the field description line for an externally described file are output indicators (positions 23 through 31), field name (positions 32 through 37), and blank after (position 39). Edit words and edit codes for fields written to an externally described file are specified in the data description specifications (DDS) for the file. Device-dependent functions such as fetch overflow (position 16) or space/skip (positions 17-22) are not valid in RPG/400 for externally described files. The overflow indicator is not valid for externally described files either. For a description of how to specify editing in the DDS, see the *Programming: Data Description Specifications Reference*.

If output specifications are used for an externally described file, the record format name is specified in positions 7 through 14 instead of the file name.

If all the fields in an externally described file are to be placed in the output record, enter *ALL in positions 32 through 37 of the field description line. If *ALL is specified, no other field description lines can be specified for that record.

If you want to place only certain fields in the output record, enter the field name in positions 32 through 37. The field names you specify in these positions must be the field names defined in the external record description, unless the field was renamed on the input specifications (see Figure 6-7 on page 6-16).

You should know about these considerations for using the output specifications for an externally described file:

- In the output of an update record, only those fields specified in the output field specifications and meeting the conditions specified by the output indicators are placed in the output record to be rewritten. The fields not specified in the output specifications are rewritten using the values that were read. This technique offers a good method of control as opposed to the UPDAT operation code that updates all fields.
- In the creation of a new record, the fields specified in the output field specifications are placed in the record. Those fields not specified in the output field specifications or not meeting the conditions specified by the output indicators are written as zeros or blanks depending on the data format specified in the external description.

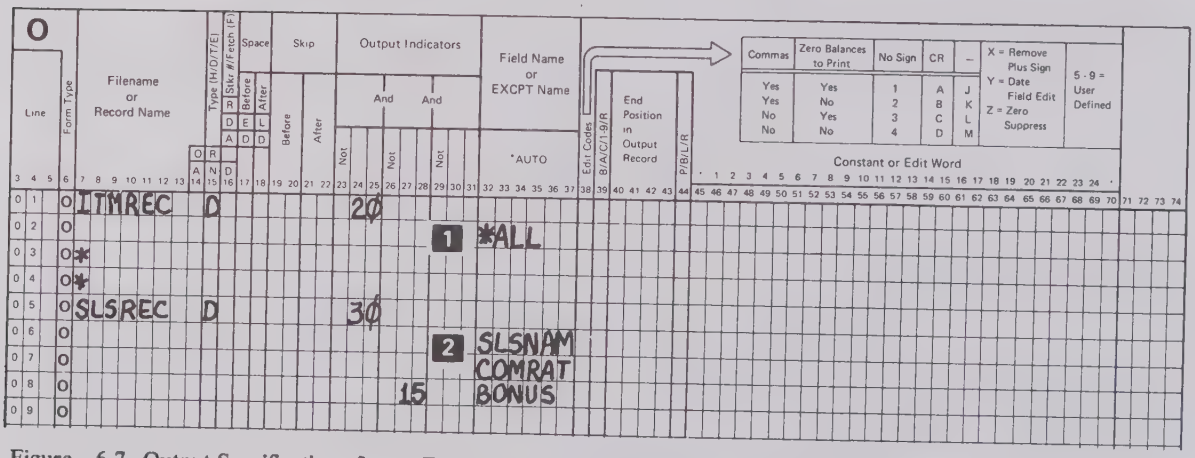


Figure 6-7. Output Specifications for an Externally Described File

1 For an update file, all fields in the record are written to the externally described record ITMREC using the current values in the program for all fields in the record.

For the creation of a new record, all fields in the record are written to the externally described record ITMREC using the current values in the program for the fields in the record.

2 To update a record, the fields SLSNAM and COMRAT are written to the externally described record SLSREC when indicator 30 is on. The field BONUS is written to the SLSREC record when indicators 30 and 15 are on. All other fields in the record are written with the values that were read.

To create a new record, the fields SLSNAM and COMRAT are written to the externally described record SLSREC when indicator 30 is on. The field BONUS is written when indicators 30 and 15 are on. All other fields in the record are

written as zeros or blanks, depending on whether the field is numeric or character.

Program Described Files

Program described files are files whose records and fields are described on input/output specifications in the program that uses the file.

Figure 6-8 shows how to specify sequence checking when your input data must contain exactly one record of the first type (01 in positions 15 and 16), followed by at least one record of another type (02 through 04 in positions 15 and 16) in each group of records read. When the specifications shown in Figure 6-8 are used and two consecutive records of the first type are read, an run-time error occurs.

If each group of input records must contain exactly one record of a particular type, but that record need not be followed by any records of other types, then specify no sequence checking (alphabetic entry in positions 15 and 16). In this case, no effective sequence checking can be carried out.

Write operations to a program described file require a data structure name in the result field.

I

Line	Form Type	Filename	Sequence	Number (1/N)	Option (O), U, S	Record Identifying Indicator or DS	Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location: From	Field Location: To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Field Indicators: Minus	Field Indicators: Zero or Blank
			O A	R N	D																									
3 4 5	6	7 8 9 10 11 12 13	14	15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70
01	I	INPUT																												
02	I		01	LL	10			L		CA																				
03	I																				2	60		TYPE						
04	I		02	N0	20			L		CB											2	LL		KEY						
05	I																													
06	I		03	N0	30			L		CC											2	21		NAME						
07	I																													
08	I		04	N0	40			L		CD											2	6		NUMBER						
09	I																													
10	I																													
11	I																													

Figure 6-8. Input Specifications for Sequence Checking

Printer Files

The PRINTER file allows you to print the output file to be printed. A maximum of eight printer files is allowed per program. PRINTER must be assigned as the device name for the file, and each file must have a unique file name. The CL command CRTPRTF (Create Print File) can be used to create a printer file (see the *Programming: Control Language Reference* for further information on the CRTPRTF command); or

the IBM-supplied file names can be used (see the *Programming: Data Management Guide* for more information on these file names).

The file operation codes that are valid for a PRINTER file are WRITE, OPEN, CLOSE, and FEOD. For a complete description of these operation codes, see Languages: RPG/400 Reference.

PRINTER files can be either externally described or program described. Overflow indicators OA-OG and OV, fetch overflow, space/skip entries, and the PRTCTL option are invalid for an externally described PRINTER file. See the *Languages: RPG/400 Reference* for the valid output specification entries for an externally described file. See the *Programming: Data Description Specifications Reference* for information on the data description specifications (DDS) for externally described printer files.

For an externally described PRINTER file, you can specify the DDS keyword INDARA. You will get a run-time error if you try to use this keyword for a program described PRINTER file.

Page Overflow

An important consideration when using a PRINTER file is page overflow. For an externally described PRINTER file, you are responsible for handling page overflow. Do one of the following:

- Count the number of output lines per page.
- Check for a file exception/error by specifying an indicator in positions 56 and 57 of the calculation specifications that specify the output operation, or by specifying a file exception/error subroutine (INFSR) that can handle the error. The file information data structure (INFDS) has detailed information on the file exception/error. See "Exception/Error Handling" on page 4-35 for further information on exception and error handling.
- Specify an indicator 01 through 99 as the overflow indicator in positions 33 and 34 of the file description specifications.
- Check INFDS for line number and page overflow (Refer to "File Information Data Structure" on page 4-23).

When a record format requires more than one line, the complete record format is printed. You should allow space for the longest record format when you specify the overflow line number in positions 20 through 22 of the line counter specifications.

For either a program described or an externally described file, you may specify an indicator 01 through 99 in positions 33 and 34 of the file description specifications. This indicator is set on when a line is printed on the overflow line, or the overflow line is reached or passed during a space or skip operation. Use the indicator to condition your response to the overflow condition. The indicator does not condition the RPG/400 overflow logic as an overflow indicator (OA through OG, OV) does. You are responsible for setting the indicator off.

For a program described PRINTER file, the following sections on overflow indicators and fetch overflow logic apply.

Overflow Indicators

An overflow indicator (0A through 0G, 0V) is set on when the last line on a page has been printed or passed. An overflow indicator can be used to specify which lines are to be printed on the next page. Overflow indicators can be specified only for program described PRINTER files and are used primarily to condition the printing of heading lines. An overflow indicator is defined in positions 33 and 34 of the file description specifications and can be used to condition operations in the calculation specifications (positions 9 through 17) and output specifications (positions 23 through 31). If an overflow indicator is not specified, RPG/400 assigns the first unused overflow indicator to the PRINTER file. Overflow indicators can also be specified as resulting indicators on the calculation specifications (positions 54 through 59).

RPG/400 sets on an overflow indicator only the first time an overflow condition occurs on a page. An overflow condition exists whenever one of the following occurs:

- A line is printed past the overflow line.
- The overflow line is passed during a space operation.
- The overflow line is passed during a skip operation.

Table 6-1 on page 6-20 shows the results of the presence or absence of an overflow indicator on the file description and/or output specifications.

The following considerations apply to overflow indicators used on the output specifications:

- Spacing past the overflow line sets the overflow indicator on.
- Skipping past the overflow line to any line on the same page sets the overflow indicator on.
- Skipping past the overflow line to any line on the new page does not set the overflow indicator on unless a skip-to is specified past the specified overflow line.
- A skip to a new page specified on a line not conditioned by an overflow indicator sets the overflow indicator off before the forms advance to a new page.
- If you specify a skip to a new line and the printer is currently on that line, a skip does not occur. Also, the overflow indicator is set to off, unless the line is past the overflow line.
- When an OR line is specified for an output print record, the space and skip entries of the preceding line are used. If they differ from the preceding line, enter space and skip entries on the OR line.
- Control level indicators can be used with an overflow indicator so that each page will contain information from only one control group (see Figure 6-9 on page 6-21).
- For conditioning an overflow line, an overflow indicator can appear in either an AND or an OR relationship. For an AND relationship, the overflow indicator must appear on the main specification line in order for that line to be considered an overflow line. For an OR relationship, the overflow indicator can be specified on

either the main specification line or the OR line. However, only one overflow indicator can be associated with one group of output indicators. For an OR relationship, only the conditioning indicators on the specification line where an overflow indicator is specified will be used for the conditioning of the overflow line.

- If an overflow indicator is used on an AND line, the line is *not* an overflow line. In this case, the overflow indicator is treated like any other output indicator.
- When the overflow indicator is used in an AND relationship with a record identifying indicator, unusual results are often obtained because the record type might not be the one read when overflow occurred. Therefore, the record identifying indicator is not on, and all lines conditioned by both overflow and record identifying indicators do not print.
- An overflow indicator can condition an exception line (E in position 15), and can condition fields within the exception record.

Table 6-1. Results of the Presence or Absence of an Overflow Indicator

File Description Specifications Positions 33-34	Output Specifications Positions 23-31	Action
No entry	No entry	First unused overflow indicator used to condition skip to next page at overflow.
No entry	Entry	Error at compile time; overflow indicator dropped from output specifications. First unused overflow indicator used to condition skip to next page at overflow.
Entry	No entry	Continuous printing; no overflow recognized.
Entry	Entry	Processes normal overflow.

Below is an example of the coding necessary for printing headings on every page: first page, every overflow page, and each new page to be started because of a change in control fields (L2 is on). Line 01 allows the headings to be printed at the top of a new page (skip to 06) only when an overflow occurs (OA is on and L2 is not on).

Line 02 allows printing of headings on the new page only at the beginning of a new control group (L2 is on). This way, duplicate headings caused by both L2 and OA being on at the same time do not occur. Line 02 allows headings to be printed on the first page after the first record is read because the first record always causes a control break (L2 turns on) if control fields are specified on the record.

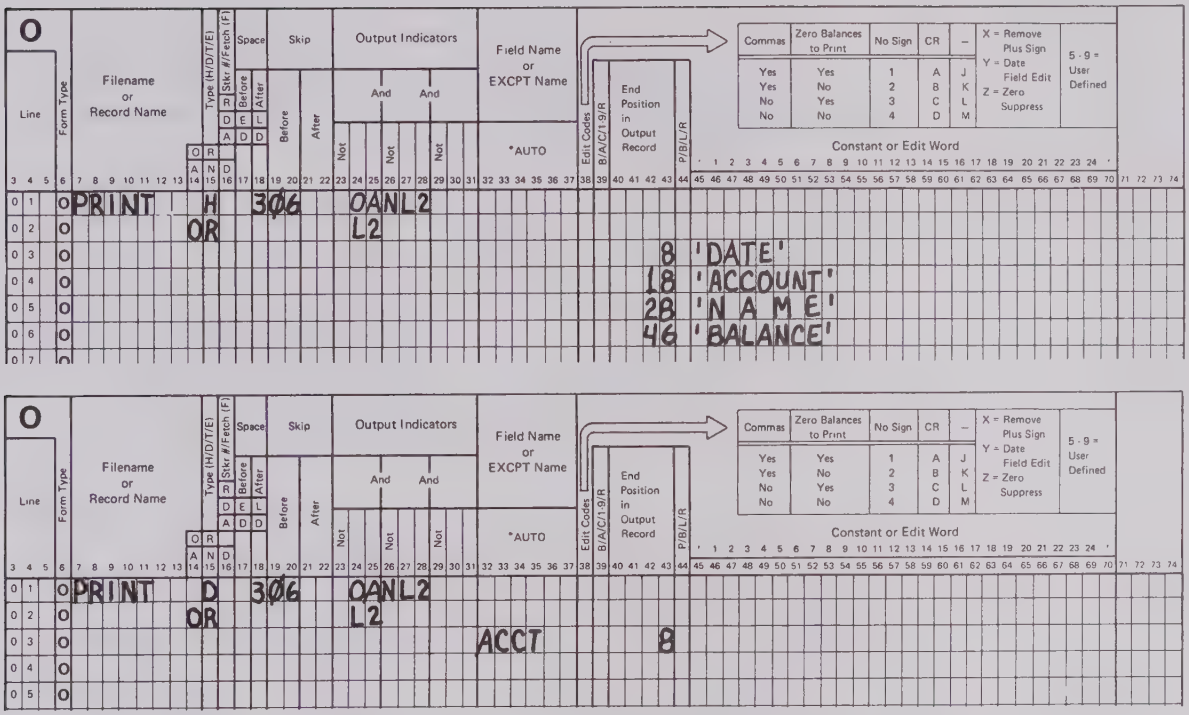


Figure 6-9. Using Control Level Indicators with an Overflow Indicator

This is the necessary coding for the printing of certain fields on every page; a skip to 06 is done either on an overflow condition or on a change in control level (L2). The NL2 indicator in line 01 prevents the line from printing and skipping twice in the same cycle.

Fetch Overflow Logic

The fetch overflow routine allows you to alter the basic RPG/400 overflow logic to prevent printing over the perforation and to let you use as much of the page as possible. During the regular program cycle, RPG/400 checks only once, immediately after total output, to see if the overflow indicator is on. When the fetch overflow function is specified, RPG/400 checks overflow on each line for which fetch overflow is specified (see Figure 6-10 on page 6-23).

Specify fetch overflow with an F in position 16 of the output specifications on any detail, total, or exception lines for a PRINTER file. The fetch overflow routine does not automatically cause forms to advance to the next page.

During output, the conditioning indicators on an output line are tested to determine whether the line is to be written. If the line is to be written and an F is specified in position 16, RPG/400 tests to determine whether the overflow indicator is on. If the overflow indicator is on, the overflow routine is fetched and the following operations occur:

1. Only the overflow lines for the file with the fetch specified are checked for output.
2. All total lines conditioned by the overflow indicator are written.

3. Forms advance to a new page when a skip to a line number less than the line number the printer is currently on is specified in a line conditioned by an overflow indicator.
4. Heading, detail, and exception lines conditioned by the overflow indicator are written.
5. The line that fetched the overflow routine is written.
6. Any detail and total lines left to be written for that program cycle are written.

Position 16 of each OR line must contain an F if the overflow routine is to be used for each record in the OR relationship. Fetch overflow cannot be used if an overflow indicator is specified in positions 23 through 31 of the same specification line. If this occurs, the overflow routine is not fetched.

Using the fetch overflow routine when printing a particular line causes overflow and there is not enough space left on the page to print the remaining detail, total, exception, and heading lines conditioned by the overflow indicator. To determine when to fetch the overflow routine, study all possible overflow situations. By counting lines and spaces, you can calculate what happens if overflow occurs on each detail, total, and exception line.

Figure 6-11 on page 6-24 shows the use of fetch overflow.

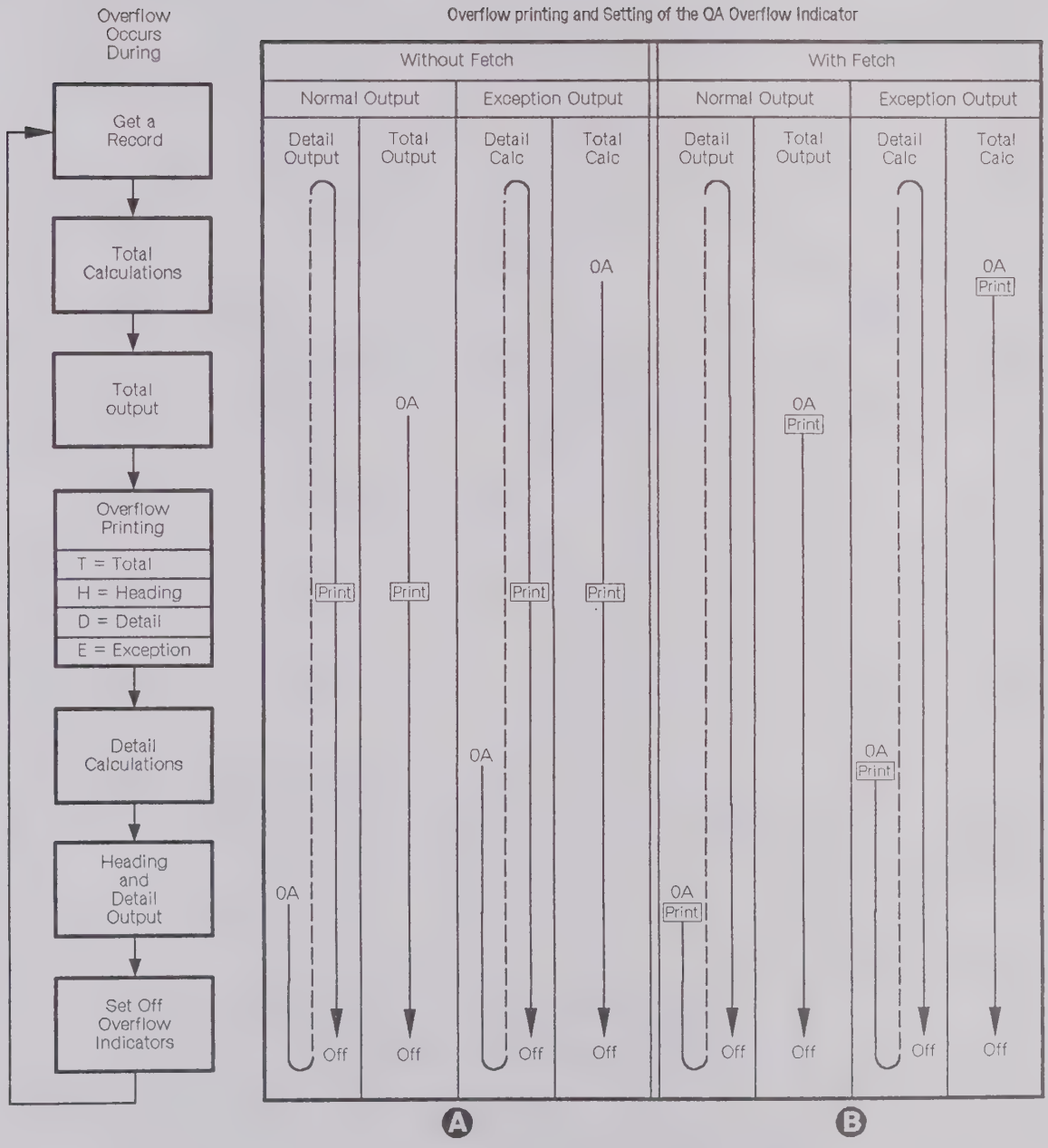
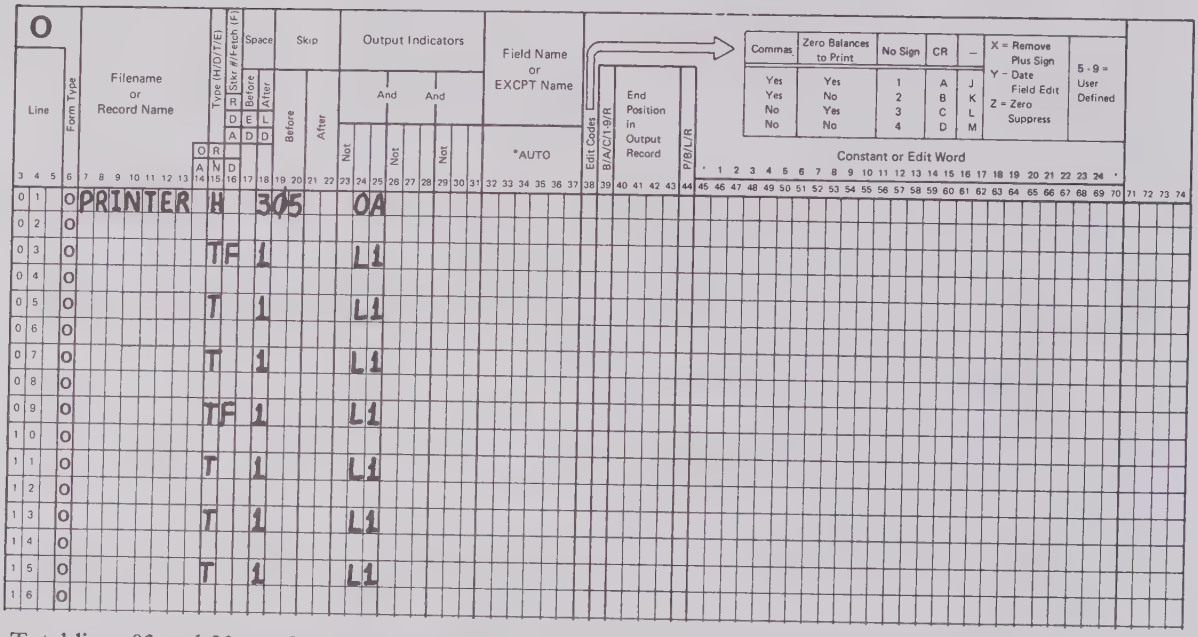


Figure 6-10. Overflow Printing: Setting of the Overflow Indicator

A When fetch overflow is not specified, the overflow lines print after total output. No matter when overflow occurs (0A is on), the overflow indicator 0A remains on through overflow output time and is set off after heading and detail output time.

B When fetch overflow is specified, the overflow lines are written before the output line for which fetch overflow was specified, if the overflow indicator 0A is on. When 0A is set on, it remains on until after heading and detail output time. The overflow lines are not written a second time at overflow output time unless overflow is sensed again since the last time the overflow lines were written.



Total lines 03 and 09 can fetch the overflow routine. However, they only do this if overflow is sensed prior to the printing of one of these lines. Before the operation in line 03 is processed, a check is made to determine whether the overflow indicator is on. If it is on, the overflow routine is fetched, the heading line conditioned by the overflow indicator is printed, and the following operations (lines 03, 05, 07, 09, 11, 13, and 15) are processed.

Figure 6-11. Use of Fetch Overflow

PRTCTL (Printer Control) Option

The PRTCTL (printer control) option allows you to modify forms control information and to access the current line value within the program for a program described PRINTER file.

Specify the PRTCTL option on a file description specifications continuation line for the PRINTER file with the following:

Note: If the file has a share ODP or user-controlled open, then the line count value may be incorrect.

Position	Entry
6	F

Position	Entry
7-52	Blank if the information is specified on a separate continuation line
53	K (indicates a continuation line)
54-59	PRTCTL
60-65	Name of the data structure used to contain the printer control and line count information
66-74	Blank if the information is specified on a separate continuation line.

The data structure specified in positions 60 through 65 of the file description specifications continuation line must be specified on the input specifications and must contain at least the following five subfields specified in the following order:

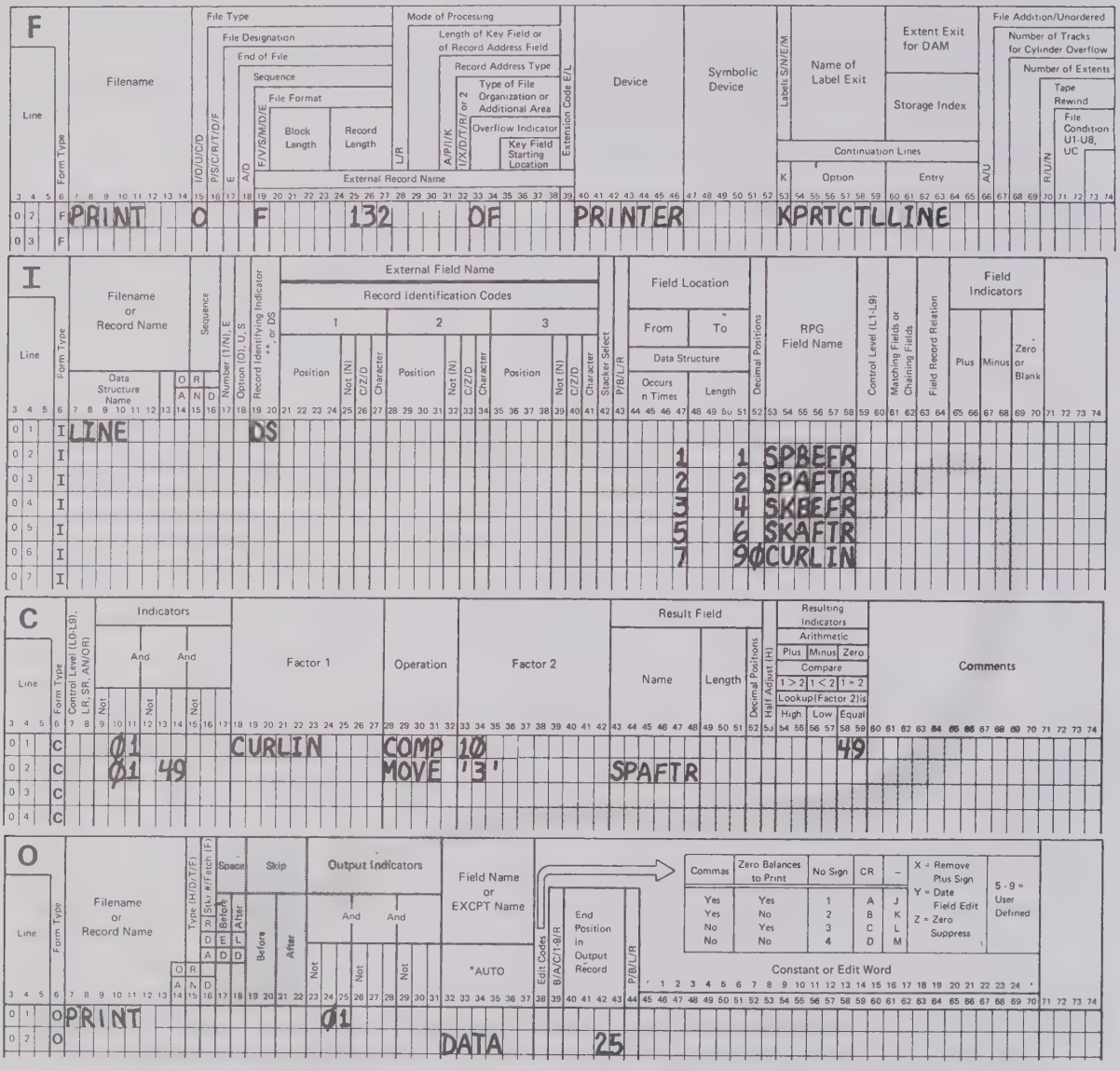
Data Structure Positions	Subfield Contents
1	A 1-position character field that contains the space-before value
2	A 1-position character field that contains the space-after value
3-4	A 2-position character field that contains the skip-before value
5-6	A 2-position character field that contains the skip-after value
7-9	A 3-digit numeric field with zero decimal positions that contains the current line count value.

The values contained in the first four subfields of the data structure are the same as those allowed in positions 17 through 22 (space and skip entries) of the output specifications. If the space/skip entries (positions 17 through 22) of the output specifications are blank, and if subfields 1 through 4 are also blank, the default is to space 1 after. If the PRTCTL option is specified, it is used only for the output records that have blanks in positions 17 through 22. You can control the space and skip value (subfields 1 through 4) for the PRINTER file by changing the values in these subfields while the program is running (see Figure 6-12 on page 6-26).

Subfield 5 contains the current line count value. RPG/400 does not initialize subfield 5 until after the first output line is printed. RPG/400 then modifies subfield 5 after each output operation to the file.

Figure 6-13 on page 6-27 is a processing chart for PRINTER files.

File Description Specifications

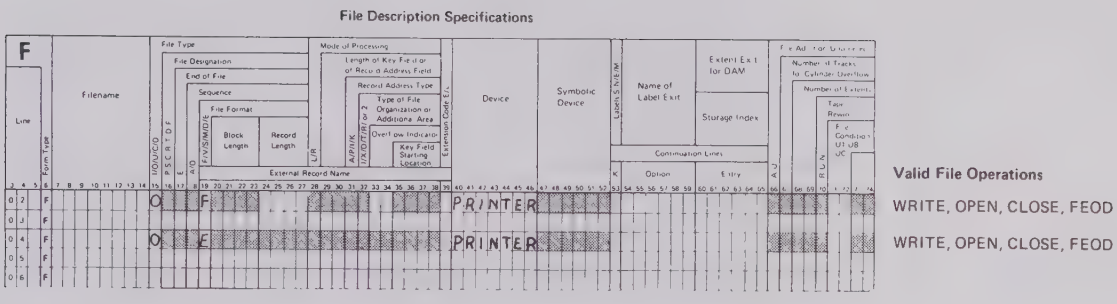


On the file description specifications, the PRTCTL option is specified for the PRINT file. The name of the associated data structure is LINE.

The LINE data structure is defined on the input specifications as having only those subfields that are predefined for the PRTCTL data structure. The first four subfields in positions 1 through 6 are used to supply space and skip information that is generally specified in positions 17 through 22 of the output specifications. The PRTCTL option allows you to change these specifications within the program.

In this example, the value in the SPAFTR subfield is changed to 3 when the value in the CURLIN (current line count value) subfield is equal to 10. (Assume that indicator 01 was set on as a record identifying indicator.)

Figure 6-12. Example of the PRTCTL Option



Note: Shaded positions must be blank, and positions without entries are program dependent.

Figure 6-13. Processing Chart for PRINTER Files

SEQ (Sequential) File

A sequential (SEQ) device specification (positions 40 through 46) in the file description specification indicates that the input or output is associated with a sequentially organized file (see Figure 6-14 on page 6-28). The actual device to be associated with the file while the running RPG/400 program can be specified by a OS/400 override command (see the *Programming: Control Language Reference* for further information), or by the file description that is pointed to by the file name.

Figure 6-15 on page 6-28 shows the operation codes allowed for a SEQ file.

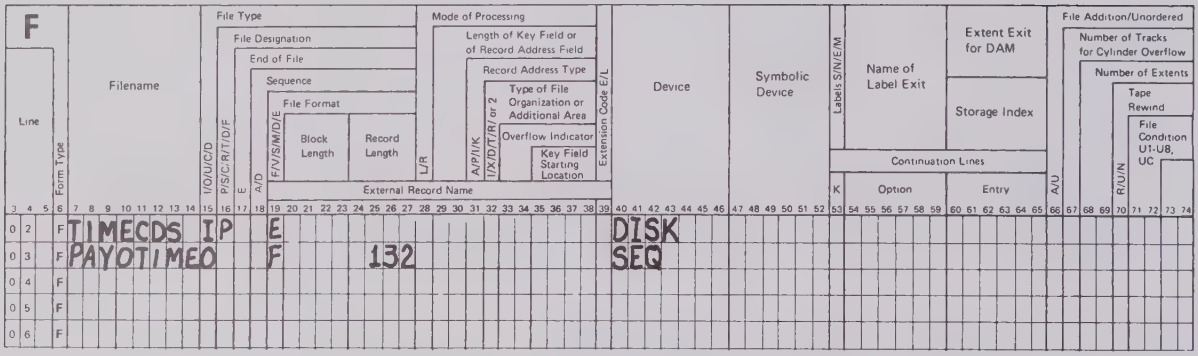
Any sequentially organized file, such as diskette, tape, data base, savefile, or printer, can be associated with the SEQ device. However, if SEQ is specified in an RPG/400 program, no device-dependent functions such as space/skip, or CHAIN can be specified.

Use the SEQ device specifications whenever you write to a tape file. To write variable-length records to a tape file, also use the RCDBLKFMT parameter of either the CL command CRTTAPF or OVRTAPF (see the *Programming: Control Language Reference)*. When you use the RCDBLKFMT parameter, the length of each record that your program writes to the tape file is determined by the highest end position specified in positions 40 through 43 of the output specifications for that record. If you do not specify an end position, RPG/400 calculates the record length from the length of the fields in the record.

Read variable-length tape records as you would read records from any sequentially organized file. Be sure the record length specified on the file description specification accommodates the longest record in the file.

Figure 6-16 on page 6-29 is a processing chart for SEQ files.

File Description Specifications



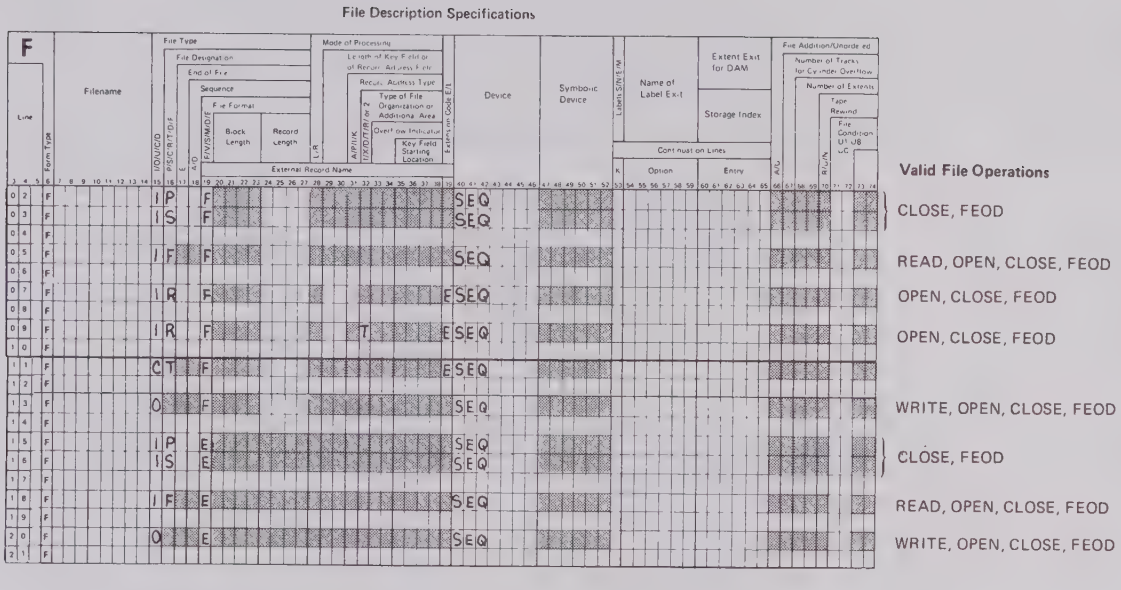
A SEQ device is specified for the PAYOTIME file. Therefore, when the program is run, a OS/400 override command can be used to specify the actual device (such as printer, tape, or diskette) that is to be associated with the file while the program is running. For example, diskette may be specified for some program runs while printer may be specified for others. The file description, which is pointed to by the file name, can specify the actual device, in which case an override command need not be used.

Figure 6-14. SEQ Device

File Description Specifications Positions 15	16	Calculation Specifications Positions 28-32
I	P/S	CLOSE, FEOD
I	F	READ, OPEN, CLOSE, FEOD
O		WRITE, OPEN, CLOSE, FEOD

Note: No print control specifications are allowed for an SEQ

Figure 6-15. Valid File Operation Codes for a SEQ



Note: Shaded positions must be blank, and positions without entries are program dependent.

Figure 6-16. Processing Chart for SEQ Files

Special File

SPECIAL in positions 40 through 46 of the file description specifications allows you to specify an input and/or output device that is not directly supported by RPG/400. The input and output operations for the file are controlled by a user-written routine. Positions 54 through 59 of the file description specifications line that contains SPECIAL in positions 40 through 46 must contain the name of the user-written routine.

RPG/400 calls this user-written routine to open the file, read and write the records, and close the file. RPG/400 creates a parameter list for use by the user-written routine. The parameter list contains an option code parameter (option), a return status parameter (status), an error-found parameter (error), and a record area parameter (area). This parameter list is accessed by RPG/400 and by the user-written routine; it cannot be accessed by the RPG/400 program that contains the SPECIAL file.

The following describes the parameters in this RPG-created parameter list:

- Option: The option parameter is a 1-position character field that indicates the action the user-written routine is to process. Depending on the operation being processed on the SPECIAL file (OPEN, CLOSE, READ, WRITE, DELET, UPDAT), one of the following values is passed to the user-written routine from RPG/400:

Value Passed	Description
O	Open the file.

Value Passed	Description
C	Close the file.
R	Read a record and place it in the area defined by the area parameter.
W	RPG/400 has placed a record in the area defined by the area parameter; it is to be written out.
D	Delete the record.
U	The record is an update of the last record read.

- Status: The status parameter is a 1-position character field that indicates the status of the user-written routine when control is returned to RPG/400. Status must contain one of the following return values when the user-written routine returns control to RPG/400:

Return Value	Description
0	Normal return. The requested action was processed.
1	The input file is at end of file, and no record has been returned. If the file is an output file, this return value is an error.
2	The requested action was not processed; error condition exists.

- Error: The error parameter is a 5-digit zoned numeric field with zero decimal positions. If the user-written routine detects an error, the error parameter contains an indication or value representing the type of error. The value is placed in the first five positions of location *RECORD in the file information data structure (INFDS) when the status parameter contains 2.
- Area: The area parameter is a character field whose length is equal to the record length associated with the SPECIAL file. This field is used to pass the record to or receive the record from the RPG/400 program.

You can add additional parameters to the RPG-created parameter list. To do this, specify PLIST in positions 54 through 59 and the name of the PLIST in positions 60 through 65 of a file description specifications continuation line for the SPECIAL file (see Figure 6-17 on page 6-31). Then use the PLIST operation in the calculation specifications to define the additional parameters.

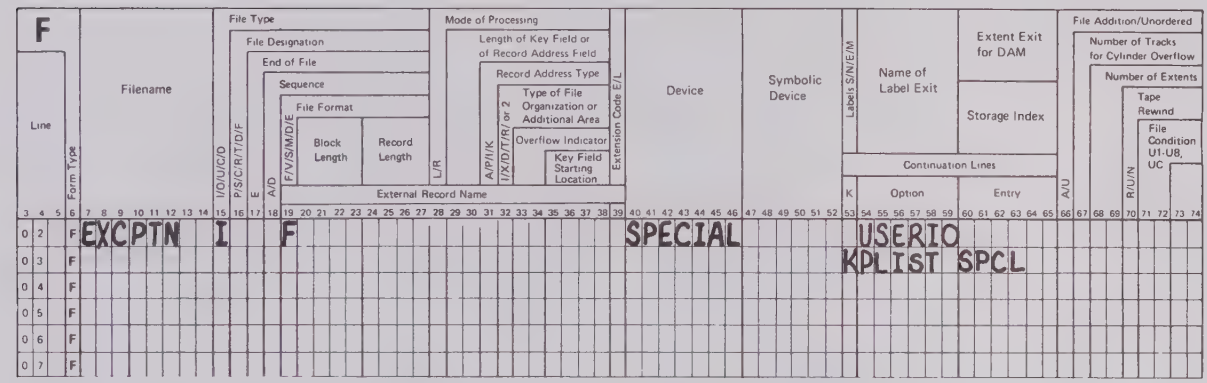
The user-written routine, the name of which is specified in positions 54 through 59 of the file description specifications for the SPECIAL file, must contain an entry parameter list that includes both the RPG-created parameters and the user-specified parameters.

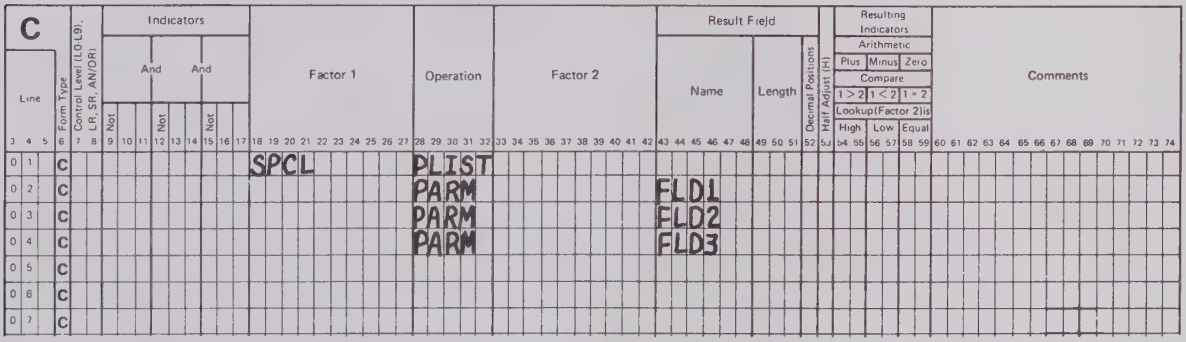
If the SPECIAL file is specified as a primary file, the user-specified parameters must be initialized before the first primary read. This can be done with a factor 2 entry on the PARM statements or by the specification of a compile-time array or an array element as a parameter.

Figure 6-18 on page 6-32 shows the file operation codes that are valid for a SPECIAL file.

Figure 6-19 on page 6-33 is a processing chart for SPECIAL files.

File Description Specifications





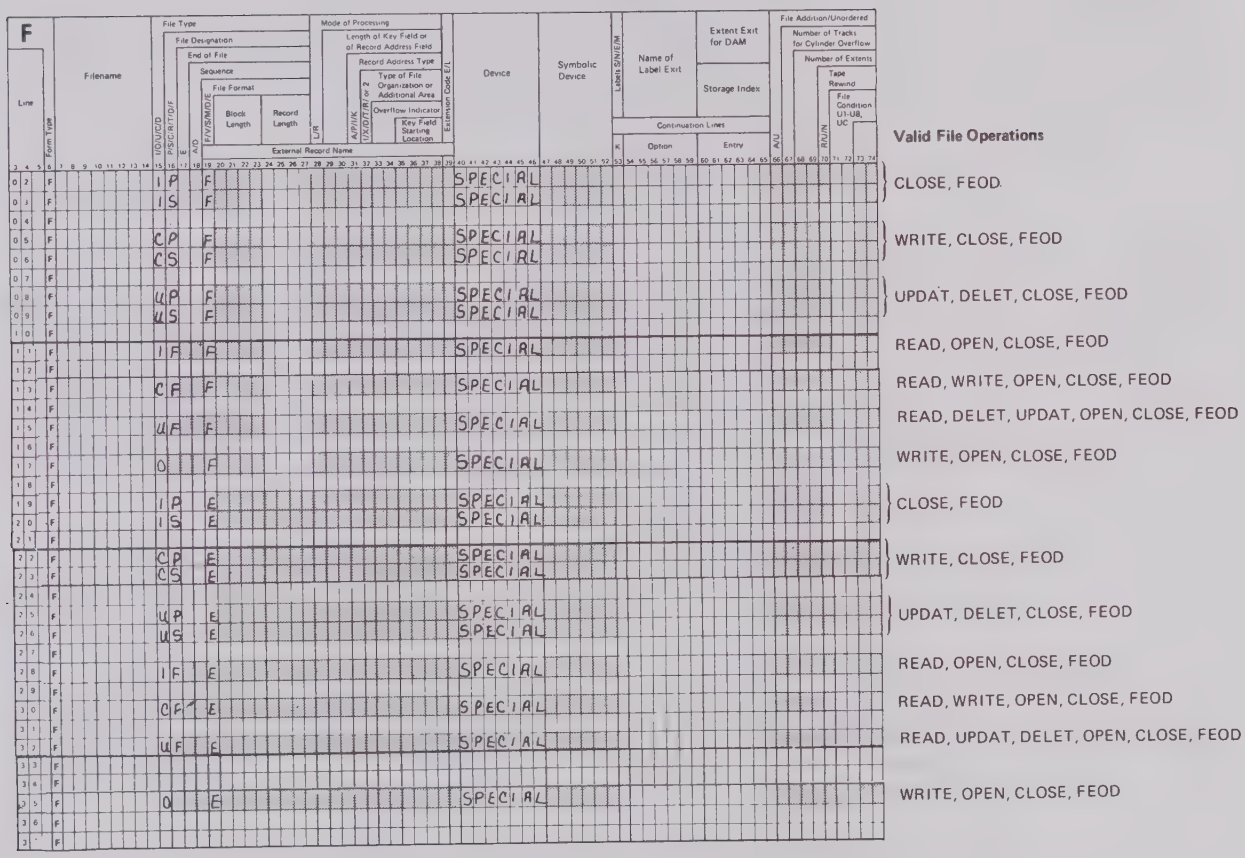
The file EXCPTN is assigned to the device SPECIAL. The I/O operations for the SPECIAL device are controlled by the user-written routine USERIO. The parameters specified for the programmer-defined PLIST SPCL are added to the end of the RPG-created parameter list for the SPECIAL device. The programmer-specified parameters can be accessed by the user RPG/400 program and the user-written routine; whereas the RPG-created parameter list can be accessed only by internal RPG/400 logic and the user-written routine.

Figure 6-17. SPECIAL Device

File Description Specifications Positions 15	16	Calculation Specifications Positions 28-32
I	P/S	CLOSE, FEOD
C	P/S	WRITE, CLOSE, FEOD
U	P/S	UPDAT, DELET, CLOSE FEOD
O		WRITE, OPEN, CLOSE, FEOD
I	F	READ, WRITE, OPEN, CLOSE, FEOD
C	F	READ, WRITE, OPEN, CLOSE, FEOD
U	F	READ, UPDAT, DELET, OPEN, CLOSE, FEOD

Figure 6-18. Valid File Operations for a SPECIAL File

File Description Specifications



Notes:

1. Shaded positions must be blank, and positions without entries are program dependent.
2. Positions 54 through 59 must contain the name of the user-written routine that controls the input/output operations for the file.

Figure 6-19. Processing Chart for SPECIAL Files

Primary/Secondary Multifile Processing

In an RPG/400 program, the processing of a primary input file and one or more secondary input files, with or without match fields, is termed multifile processing. Selection of records from more than one file based on the contents of match fields is known as multifile processing by matching records. Multifile processing can be used

with externally described or program described input files that are designated as primary/secondary files.

Multifile Processing with No Match Fields

When no match fields are used in multifile processing, records are selected from one file at a time. When the records from one file are all processed, the records from the next file are selected. The files are selected in this order:

1. Primary file, if specified
2. Secondary files in the order in which they are described on the file description specifications.

Multifile Processing with Match Fields

When match fields are used in multifile processing, the program selects the records for processing according to the contents of the match fields. The program reads one record from every primary/secondary input file and compares the match fields in the records. If the records are in ascending order, the program selects the record with the lowest match field. If the records are in descending order, the program selects the record with the highest match field.

When a record is selected from a file, the program reads the next record from that file. At the beginning of the next program cycle, the new record is compared with the other records in the read area that are waiting for selection, and one record is selected for processing.

Records without match fields can also be included in the files. Such records are selected for processing before records with match fields. If two or more of the records being compared have no match fields, selection of those records is determined by the priority of the files from which the records came. The priority of the files is:

1. Primary file, if specified
2. Secondary files in the order in which they are described on the file description specifications.

When the primary file record matches one or more of the secondary records, the MR (matching record) indicator is set on. The MR indicator is on for detail time processing of a matching record through the total time that follows the record. This indicator can be used to condition calculation or output operations for the record that is selected. When one of the matching records must be selected, the selection is determined by the priority of the files from which the records came.

Figure 5-3 on page 5-10 shows the logic flow of multifile processing.

A program can be written where only one input file is defined with match fields and no other input files have match fields. The files without the match fields are then processed completely according to the previously mentioned priority of files. The file with the match fields is processed last, and sequence checking occurs for that file.

Assigning Match Field Values (M1-M9)

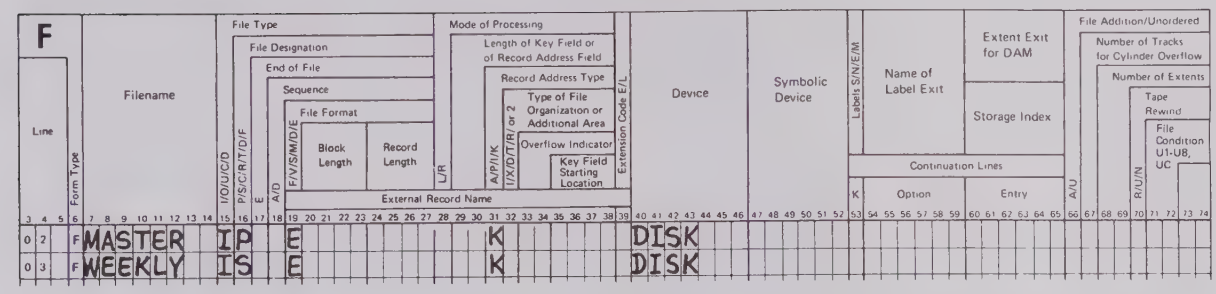
When assigning match field values (M1 through M9) to fields on the input specifications in positions 61 and 62, consider the following:

- Sequence checking is done for all record types with match field specifications. All match fields must be in the same order, either all ascending or all descending. The contents of the fields to which M1 through M9 are assigned are checked for correct sequence. An error in sequence causes the RPG/400 exception/error handling routine to receive control. When the program continues processing, the next record from the same file is read.
- Not all files used in the program must have match fields. Not all record types within one file must have match fields either. However, at least one record type from two files must have match fields if files are ever to be matched.
- The same number of match fields must be specified for all record types that are used in matching. The same matching record values must also be used for all types (see Figure 6-20 on page 6-37).
- All match fields given the same matching record value (M1 through M9) must be the same length and type (character or numeric). If the match field contains packed data, the zoned decimal length, which is (2 x the packed length) -1, is considered as the length of the match field. It is valid to match a packed field in one record against a zoned decimal field in another if the digit lengths are identical. The length must always be odd because the length of a packed field is always odd.
- Record positions of different match fields can overlap, but the total length of all fields must not exceed 256 characters.
- If more than one match field is specified for a record type, all the fields are combined and treated as one continuous field (see Figure 6-20 on page 6-37). The fields are combined according to descending sequence (M9 to M1) of matching record values.
- Match fields cannot be split; that is, the same matching field value cannot be used twice for one type of record.
- Match fields can be either character or numeric. However, all match fields given the same matching record value (M1 through M9) are considered numeric if any one of the match fields is described as numeric.
- When numeric fields having decimal positions are matched, they are treated as if they had no decimal position. For instance 3.46 is considered equal to 346.
- Only the digit portions of numeric match fields are compared. Even though a field is negative, it is considered to be positive because the sign of the numeric field is ignored. Therefore, a -5 matches a +5.
- Whenever more than one matching record value is used, all match fields must match before the MR indicator is set on. For example, if match fields M1, M2, and M3 are specified, all three fields from one record must match all three fields from the other record. A match on only the M1 and M2 fields will not set the MR indicator on (see Figure 6-20 on page 6-37).
- Matching fields cannot be used for look-ahead fields, and arrays.

- Field names are ignored in matching record operations. Therefore, fields from different record types that are assigned the same match level can have the same name.
- If an alternate collating sequence or a file translation is defined for the program, character fields are matched according to the alternate sequence specified.
- A field specified as binary (B in position 43 of the input specifications) cannot be assigned a match field value. However, a field specified as packed (P in position 43 of the input specifications) can be assigned a match field value.
- Match fields that have no field record relation indicator should be described before those that do. When the field record relation indicator is used with match fields, the field record relation indicator must be the same as a record identifying indicator for this file, and the match fields must be grouped according to the field record relation indicator. Remember that field record relation indicators are invalid for externally described files.
- When any match value (M1 through M9) is specified for a field without a field record relation indicator, all match values used must be specified once without a field record relation indicator. If all match fields are not common to all records, a dummy match field should be used (see Figure 6-21).
- Match fields are independent of control level indicators (L1 through L9).
- If multifile processing is specified and the LR indicator is set on, the program bypasses the multifile processing routine.

Figure 6-22 is an example of how match fields are specified.

File Description Specifications



The files in this example are externally described (E in position 19) and are to be processed by keys (K in position 31).

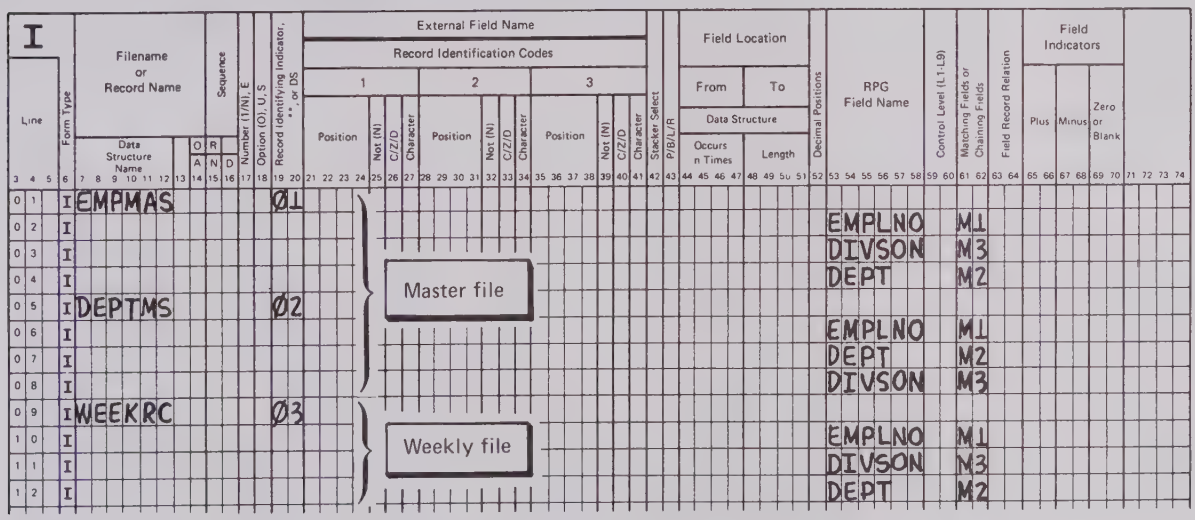


Figure 6-20. Match Fields

Because the MASTER and WEEKLY TRANSACTION files are externally described, only the field names and match level values need to be specified in the input specifications. For the same reason, record format names, rather than file names, are specified in positions 7 through 14 of the record identification lines.

Three record types are used in matching records. All record types have three match fields specified, and all use the same values (M1, M2, M3) to indicate which fields must match. The MR indicator is set on only if all three match fields in either of the record types from the MASTER file (EMPMAS and DEPTMS) are the same as all three fields from the WEEKRC record in the WEEKLY file.

The three match fields in each record type are combined and treated as one match field organized in the following descending sequence:

DIVSON	DEPT	EMPLNO
M3	M2	M1

The order in which the match fields are specified in the input specifications does not affect the organization of the match fields.

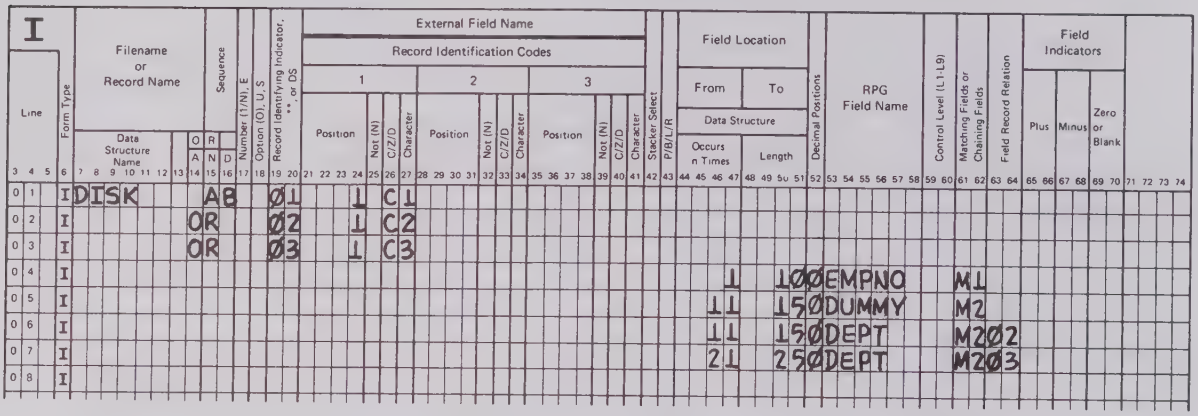


Figure 6-21 (Part 1 of 2). Dummy Match Fields

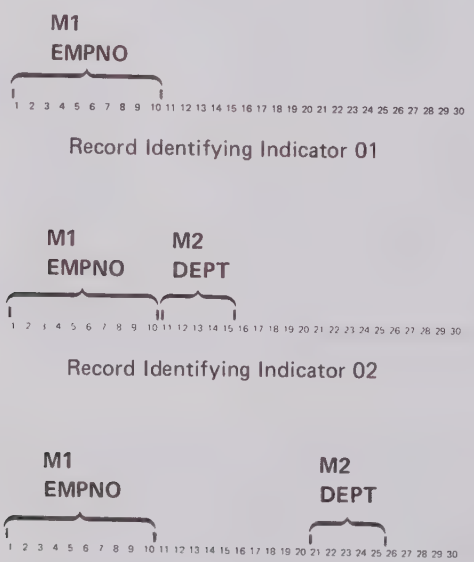


Figure 6-21 (Part 2 of 2). Dummy Match Fields

Three different record types are found in the input file. All three contain a match field in positions 1 through 10. Two of them have a second match field. Because M1 is found on all record types, it can be specified without an entry in positions 63 and 64. If one match value (M1 through M9) is specified without field record relation entries, all match values must be specified once without field record relation entries. Because the value M1 is specified without field record relationship, an M2 value must also be specified once without field record relationship. The M2 field is not on all record types; therefore a dummy M2 field must be specified next. The dummy field can be given any unique name, but its specified length must be equal to the length

of the true M2 field. The M2 field is then related to the record types on which it is found by field record relation entries (lines 06 and 07).

File Description Specifications

Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D/F	E	A/D	F/V/S/M/D/E	Block Length	Record Length	L/R	A/P/I/K	T/X/D/T/R/ or 2	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels S/N/E/M	Name of Label Exit	Extent Exit for DAM	Storage Index	K	Option	Entry	A/U	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8, UC	R/U/N
02	F	PRIMARY	I	P	E	A	F		64						DISK														
03	F	FIRSTSEC	I	S		A	F		64						DISK														
04	F	SECSEC	I	S		A	F		64						DISK														
05	F																												
06	F																												

Input Specifications

Line	Form Type	Filename or Record Name	Sequence	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, *, or DS	External Field Name / Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Data Structure: Occurs n Times	Data Structure: Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Minus	Zero or Blank
01	I	PRIMARY	AA			01		L	C	P		2	N	C																	
02	I		BB			02		L	C	P		2		C									2	3	MATCH		M1				
03	I																														
04	I																						2	3	NOM						
05	I																														
06	I	FIRSTSECAB				03		L	C	S		2	N	C																	
07	I																						2	3	MATCH		M1				
08	I		BC			04		L	C	S		2		C																	
09	I																						2	3	NOM						
10	I																														
11	I	SECSEC	AC			05		L	C	T		2	N	C																	
12	I																						2	3	MATCH		M1				
13	I		BD			06		L	C	T		2		C																	
14	I																						2	3	NOM						
15	I																														
16	I																														

Figure 6-22. Match Field Specifications for Three Disk Files

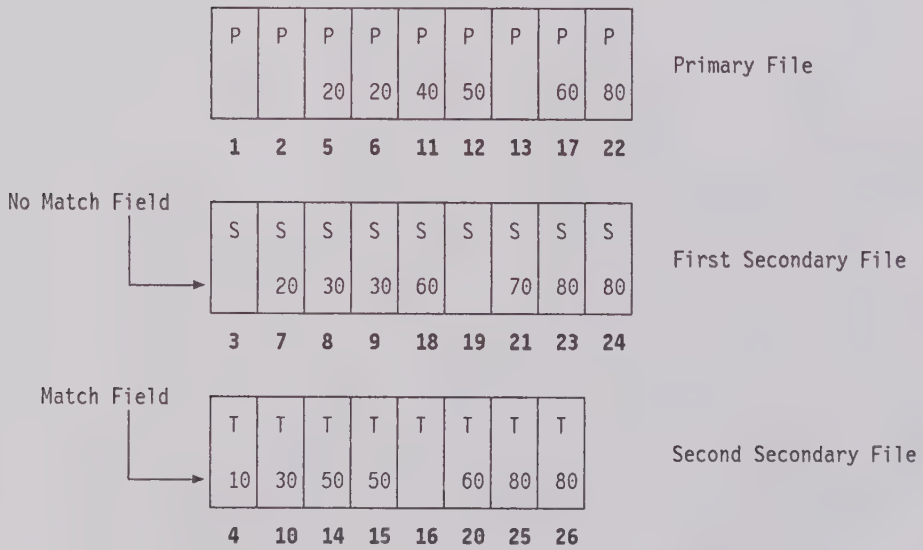
Processing Matching Records

Matching records for two or more files are processed in the following manner:

- Whenever a record from the primary file matches a record from the secondary file, the primary file is processed first. Then the matching secondary file is processed. The record identifying indicator that identifies the record type just selected is on at the time the record is processed. This indicator is often used to control the type of processing that takes place.

- Whenever records from ascending files do not match, the record having the lowest match field content is processed first. Whenever records from descending files do not match, the record having the highest match field content is processed first.
- A record type that has no match field specification is processed immediately after the record it follows. The MR indicator is off. If this record type is first in the file, it is processed first even if it is not in the primary file.
- The matching of records makes it possible to enter data from primary records into their matching secondary records because the primary record is processed before the matching secondary record. However, the transfer of data from secondary records to matching primary records can be done only when look-ahead fields are specified.

Figure 6-23 through Figure 6-24 show how records from three files are selected for processing.



The records from the three disk files above are selected in the order indicated by the dark numbers.

Figure 6-23. Normal Record Selection from Three Disk Files

Table 6-2. Normal Record Selection from Three Disk Files

Cycle	File Processed	Indicators On	Reason for Setting Indicator
1	PRIMARY	02	No match field specified
2	PRIMARY	02	No match field specified
3	FIRST SEC	04	No match field specified
4	SEC SEC	05	Second secondary low; no primary match
5	PRIMARY	01, MR	Primary matches first secondary
6	PRIMARY	01, MR	Primary matches first secondary
7	FIRST SEC	03, MR	First secondary matches primary
8	FIRST SEC	03	First secondary low; no primary match
9	FIRST SEC	03	First secondary low; no primary match
10	SEC SEC	05	Second secondary low; no primary match
11	PRIMARY	01	Primary low; no secondary match
12	PRIMARY	01, MR	Primary matches second secondary
13	PRIMARY	02	No match field specified
14	SEC SEC	05, MR	Second secondary matches primary
15	SEC SEC	05, MR	Second secondary matches primary
16	SEC SEC	06	No match field specified
17	PRIMARY	01, MR	Primary matches both secondary files
18	FIRST SEC	03, MR	First secondary matches primary
19	FIRST SEC	04	No match field specified
20	SEC SEC	05, MR	Second secondary matches primary
21	FIRST SEC	03	First secondary low; no primary match
22	PRIMARY	01, MR	Primary matches both secondary files
23	FIRST SEC	03, MR	First secondary matches primary
24	FIRST SEC	02, MR	First secondary matches primary
25	SEC SEC	05, MR	Second secondary matches primary
26	SEC SEC	05, MR	Second secondary matches primary

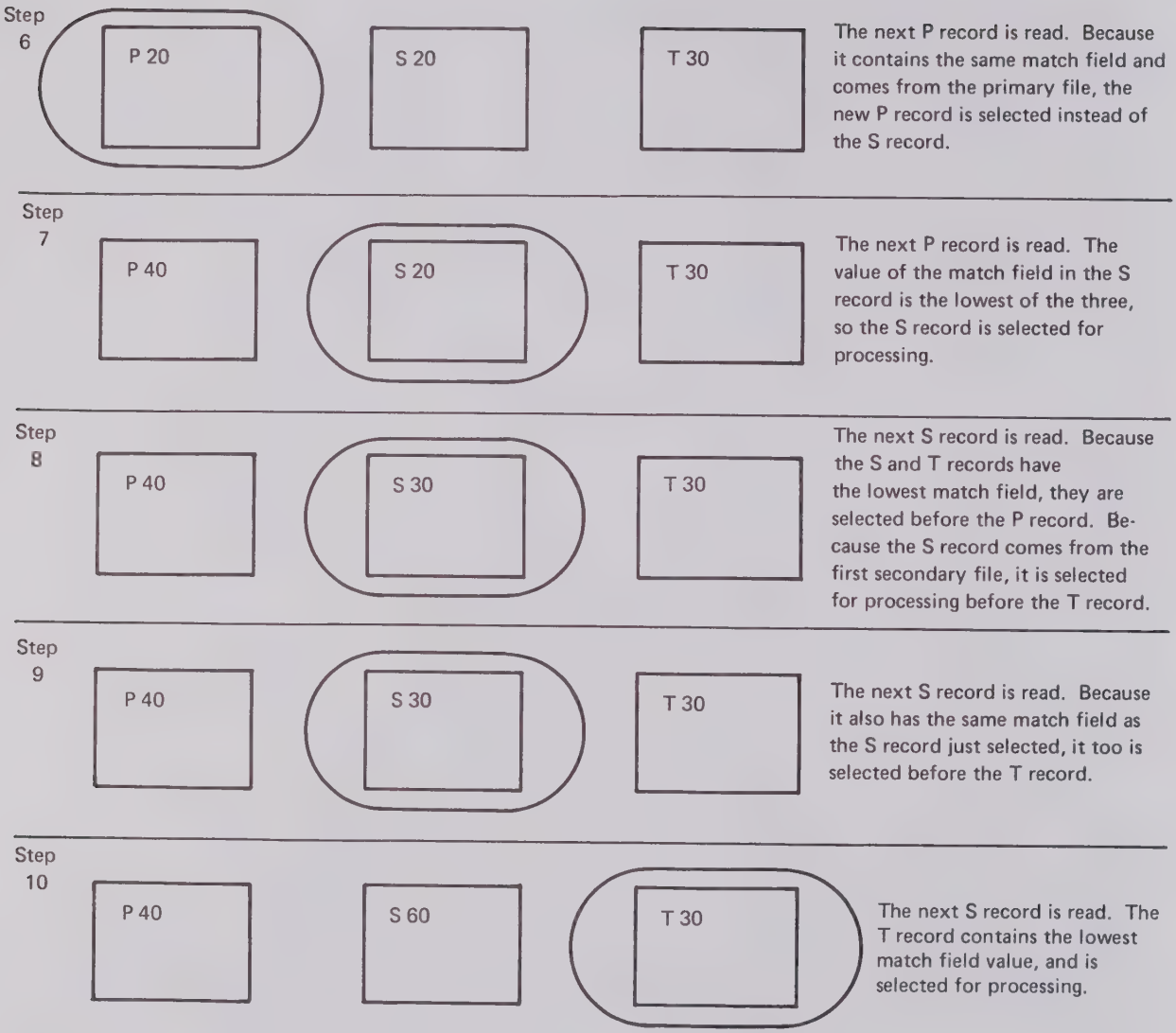


Figure 6-24 (Part 1 of 2). Normal Record Selection from Three Disk Files

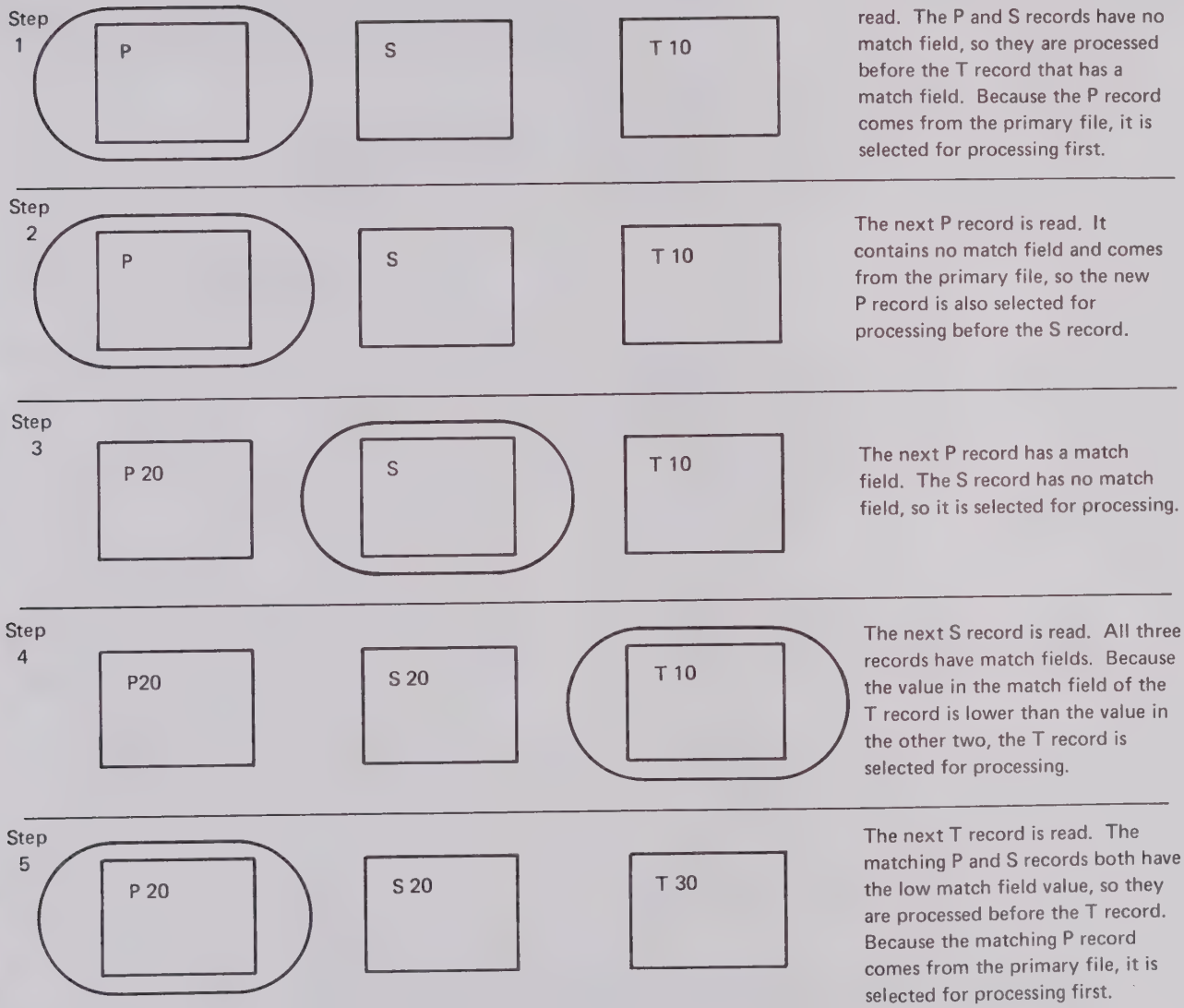


Figure 6-24 (Part 2 of 2). Normal Record Selection from Three Disk Files

Chapter 7. Commitment Control

Commitment control is a means of grouping file operations. With commitment control, you ensure one of two outcomes for the file operations in a group: either all of the file operations are successful or none of the file operations has any effect. In this way, you process a group of operations as a unit.

Using Commitment Control

To use commitment control, you do the following:

- Use the CL commands CRTJRN (Create Journal), CRTJRNRCV (Create Journal Receiver) and JRNPF (Journal Physical File) to prepare for using commitment control, and the CL commands STRCMTCTL (Start Commitment Control) and ENDCMTCTL (End Commitment Control) to notify the system when you want to start and end commitment control. See the *Programming: Control Language Reference* for information on these commands.
- Specify commitment control on the file description specifications of the files to be under commitment control.
- Use the COMIT (Commit) operation code to apply a group of changes to files under commitment control, or use the ROLBK (Roll Back) operation code to eliminate the pending group of changes to files under commitment control.

Starting and Ending Commitment Control

The CL command STRCMTCTL notifies the system that you want to process files under commitment control.

The LCKLVL (Lock Level) parameter allows you to select the level at which records are locked under commitment control. See "Commitment Control Locks" on page 7-3 in this chapter and the *Programming: Control Language Reference* for further details on lock levels.

When you complete a group of changes with a COMIT operation, you are allowed to specify a label to identify the end of the group. In the event of an abnormal job end, this identification label is written to a file, message queue or data area so that you will know which group of changes is the last group to be completed successfully. You specify this file, message queue or data area, on the STRCMTCTL command.

Issue the STRCMTCTL command before you call any program that processes files specified for commitment control. If you call a program that opens a file specified for commitment control before you issue the STRCMTCTL command, the opening of the file will fail.

The CL command ENDCMTCTL notifies the system that your routing step has finished processing files under commitment control. See the *Programming: Control Language Reference* for further information on the STRCMTCTL and ENDCMTCTL commands.

Specifying Files for Commitment Control

On the file continuation specifications, enter a K in position 53 and the word COMIT in positions 54 through 59. On the file description specifications, describe the file as having device DISK in positions 40 through 46.

When a program specifies commitment control for a file, the specification applies only to the input and output operations made by this program for this file. Commitment control does not apply to operations other than input and output operations. It does not apply to files that do not have commitment control specified in the program doing the input or output operation.

When more than one program accesses a file as a shared file, all or none of the programs must specify the file to be under commitment control.

Commitment Control Operations

The COMIT (Commit) operation tells the system that you have completed a group of changes to the files under commitment control. The ROLBK (Roll Back) operation eliminates the current group of changes to the files under commitment control. For information on how to specify these operation codes and what each operation does, see Languages: RPG/400 Reference.

If the system fails, it implicitly issues a ROLBK operation. You can check the identity of the last successfully completed group of changes using the label you specify in factor 1 of the COMIT operation code, and the notify-object you specify on the STRCMTCTL command.

At the end of a routing step, or when you issue the ENDCMTCTL command, OS/400 issues an implicit ROLBK which eliminates any changes since the last ROLBK or COMIT operation that you issued. To ensure that all your file operations have effect, issue a COMIT operation before ending a routing step operating under commitment control.

The OPEN operation permits input and output operations to be made to a file and the CLOSE operation stops input and output operations from being made to a file. However, the OPEN and CLOSE operations do not affect the COMIT and ROLBK operations. A COMIT or ROLBK operation affects a file, even after the file has been closed. For example, your program may include the following steps:

1. Issue COMIT (for files already opened under commitment control).
2. Open a file specified for commitment control.
3. Perform some input and output operations to this file.
4. Close the file.
5. Issue ROLBK.

The changes made at step 3 are rolled back by the ROLBK operation at step 5, even though the file has been closed at step 4. The ROLBK operation could be issued from another program in the same routing step.

It is not necessary for a program to operate all its files under commitment control, and to do so may adversely affect performance. The COMIT and ROLBK operations have no effect on files that are not under commitment control.

Note: When multiple devices are attached to an application program, and commitment control is in effect for the files this program uses, the COMIT or ROLBK operations continue to work on a file basis and not by device. This may result in updating the data base with partially completed COMIT blocks, or eliminating changes that other users have completed. It is the programmer's responsibility to ensure that this will not happen.

Commitment Control Locks

On the STRCMTCTL command, you specify a level of locking, either LCKLVL (*ALL) or LCKLVL (*CHG). When your program is operating under commitment control and has processed an input or output operation on a record in a file under commitment control, the record is locked as follows:

- Your program may access the record.
- Another program in your routing step, with this file under commitment control, may read the record. If the file is a shared file, the second program may also update the record.
- Another program in your routing step that does not have this file under commitment control cannot read or update the record.
- Another program in a separate routing step, with this file under commitment control, may read the record if you specified LCKLVL (*CHG), but it cannot read the record if you specified LCKLVL (*ALL). With either lock level, the second program cannot update the record.
- Another program that does not have this file under commitment control and that is not in your routing step can read but not update the record.

The COMIT and ROLBK operations release the locks on the records. See the *Programming: Control Language Reference* for details on lock levels.

Note: The SETLL and SETGT operations will lock a record in the same cases where a read operation (not for update) would lock a record for commitment control.

Commitment Control in the Program Cycle

Commitment control is intended for full procedural files, where the input and output is under the control of the programmer. We recommend you do not use commitment control with primary and secondary files, where input and output is under the control of the RPG/400 program cycle; here are some of the reasons why:

- You cannot issue a COMIT operation for the last total output in your program.
- It is difficult to program within the cycle for recovery from a locked-record condition.
- Level indicators are not reset by the ROLBK operation.
- After a ROLBK operation, processing matching records may produce a sequence error.

Example of Using Commitment Control

Here is an example of the specifications and CL commands for a program operating under commitment control.

To prepare for using commitment control, you issue the following CL commands:

- `CRTJRNRCV JRNRCV (RECEIVER)`

  This command creates a journal receiver named `RECEIVER`.

- `CRTJRN JRN(JOURNAL) JRNRCV(RECEIVER)`

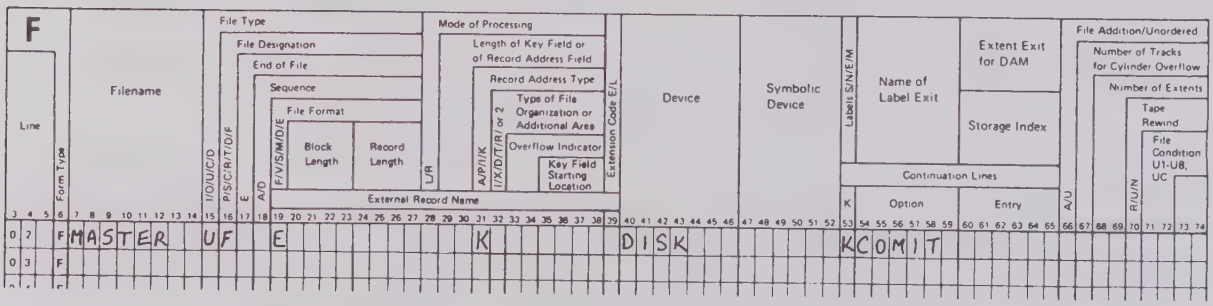
  This command creates a journal named `JOURNAL` and attaches the journal receiver named `RECEIVER`.

- `JRNPF FILE(MASTER) JRN(JOURNAL)`

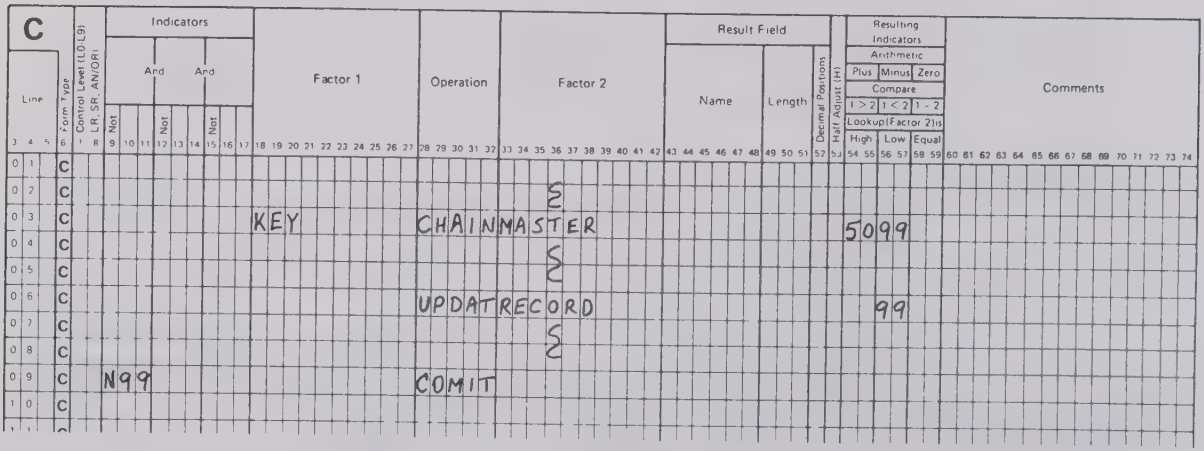
  This command directs journal entries for the file `MASTER` to the journal `JOURNAL`.

In your program, you specify `COMIT` for the file `MASTER:`.

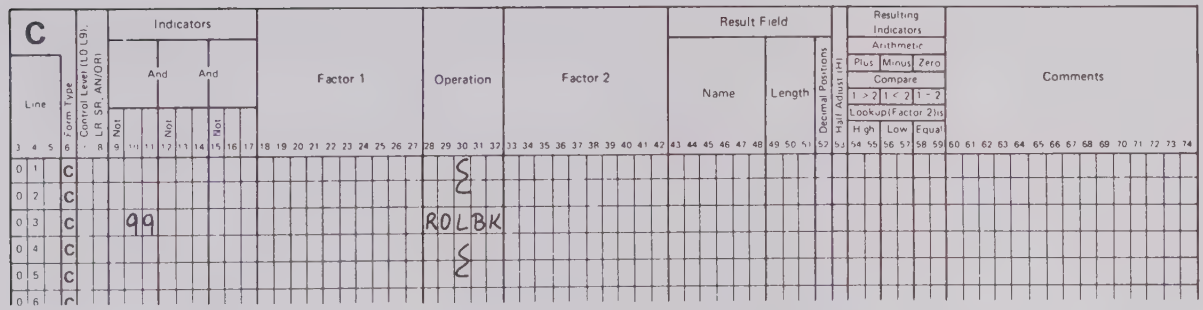
File Description Specifications



In the calculation specifications, you use the `COMIT` operation to complete a group of operations:



If an operation within a group fails, you may use the `ROLBK` operation to eliminate the entire group of operations:



To operate your program (named REVISE) under commitment control, you issue the commands:

- STRCMTCTL LCKLVL(*ALL)

  This command starts commitment control, with the highest level of locking.

- CALL REVISE

  This command calls your program (named REVISE).

- ENDCMTCTL

  This command ends commitment control, and causes an implicit Roll Back operation.

Chapter 8. Using DISK Files

Data base files, which are associated with the RPG/400 device DISK in positions 40 through 46 of the file description specifications, can be:

- Externally described files, whose fields are described to the OS/400 through data description specifications (DDS)
- Program described files, whose fields are described on input/output specifications in the program that uses the file.

All data base files are created by OS/400 create file commands. See the *Programming: Control Language Reference* for a description of the OS/400 commands that relate to data base files.

Externally Described Disk Files

Externally described DISK files are identified by an E in position 19 of the file description specifications. The E indicates that the compiler is to retrieve the external description of the file from the system when the program is compiled. Therefore, the file must be created before the program is compiled.

The external description for a DISK file includes:

- The record format specifications that contain a description of the fields in a record
- Access path specifications that describe how the records are to be retrieved.

These specifications result from the DDS for the file and the OS/400 create file command that is used for the file.

Record Format Specifications

The record format specifications allow you to describe the fields in a record and the location of the fields in a record. The fields are located in the record in the order specified in DDS. The field description generally includes the field name, the field type (character, binary, zoned decimal, or packed decimal), and the field length (including the number of decimal positions in a numeric field). Instead of being specified in the record format for a physical or logical file, you can define the field attributes in a field reference file (see Figure 8-1 and Figure 8-2).

In addition, DDS keywords can be used to:

- Specify that duplicate key values are not allowed for the file (UNIQUE)
- Specify a text description for a record format or a field (TEXT).

For a complete list of the DDS keywords that are valid for a data base file, see *Programming: Data Base Guide.*

Figure 8-1 on page 8-3 and Figure 8-2 on page 8-4 show an example of the DDS for a data base file and for a field reference file that defines the attributes for the

fields used in the data base file. See the *Programming: Data Description Specifications Reference* for more information on a field reference file.

Access Path

The description of an externally described file contains the access path that describes how records are to be retrieved from the file. Records can be retrieved based on an arrival sequence (nonkeyed) access path or on a keyed sequence access path.

The arrival sequence access path is based on the order in which the records are stored in the file. Records are added to the file one after another.

For the keyed sequence access path, the sequence of records in the file is based on the contents of the key field that is defined in the DDS for the file. For example, in the DDS shown in Figure 8-1 on page 8-3, CUST is defined as the key field. The keyed sequence access path is updated whenever records are added, deleted, or the contents of a key field change.

For a complete description of the access paths for an externally described data base file, see the *Programming: Data Base Guide*.

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754- UM/050*
*Printed in U.S.A.

File | Keying Instruction | Graphic / Key | Description | Page of
Programmer | Date

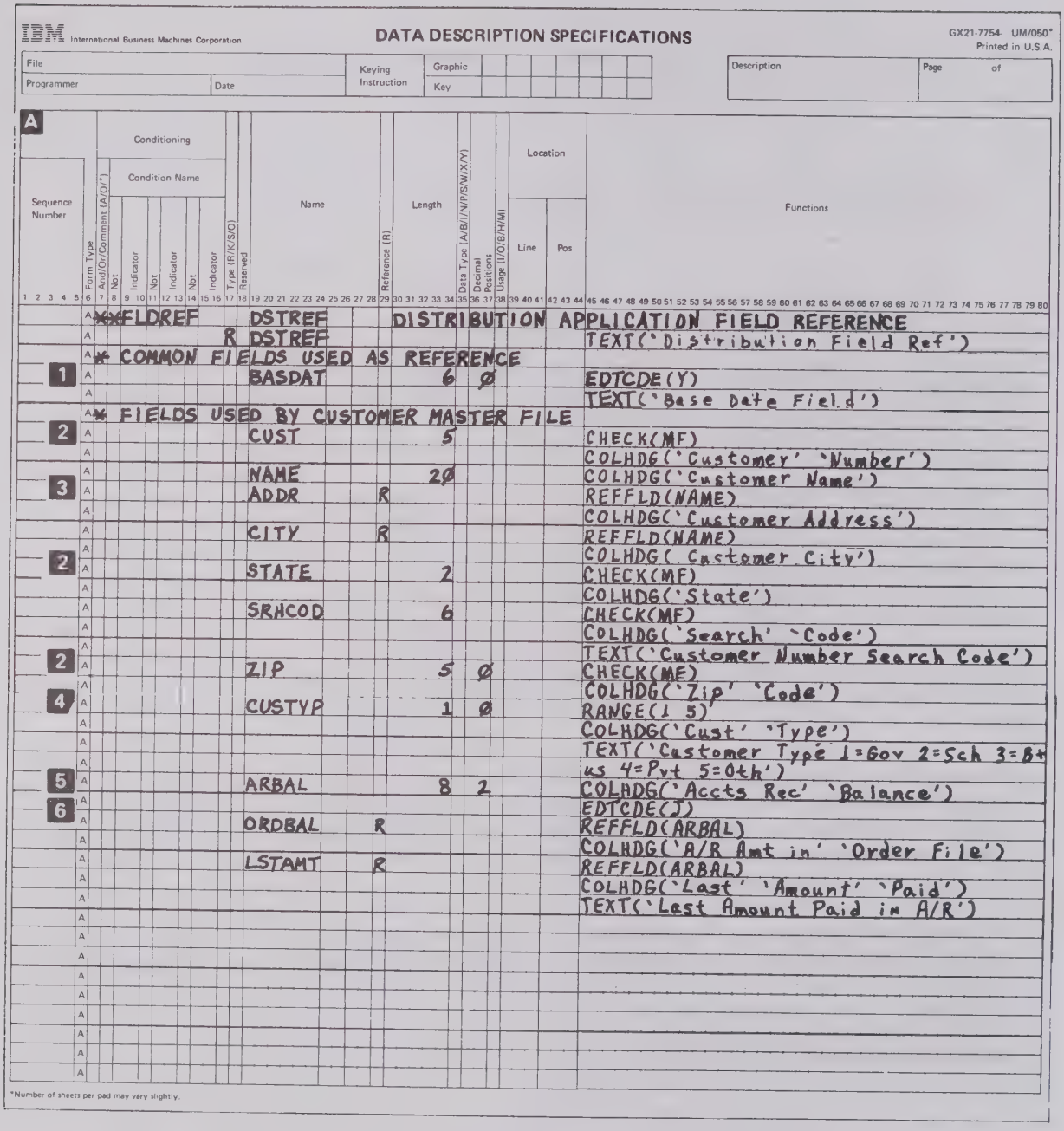
A

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Condition Name: Not	Indicator	Not	Indicator	Not	Indicator	Type (R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (A/B/I/P/S/W/X/Y)	Decimal Positions	Usage (I/O/B/H/M)	Location: Line	Location: Pos	Functions
	A	**									CUSMSTL								LOGICAL CUSTOMER MASTER FILE
	A																		UNIQUE
	A								R		CUSREC								PFILE(CUSMSTP)
	A																		TEXT('Customer Master Record')
	A										CUST								
	A										NAME								
	A										ADDR								
	A										CITY								
	A										STATE								
	A										ZIP								
	A										SRHCOD								
	A										CUSTYP								
	A										ARBAL								
	A										ORDBAL								
	A										LSTAMT								
	A										LSTDAT								
	A										CRDLMT								
	A										SLSYR								
	A										SLSLYR								
	A								K		CUST								

The sample data description specifications (DDS) are for the customer master logical file CUSMSTL. The file contains one record format CUSREC (customer master record). The data for this file is contained in the physical file CUSMSTP, which is identified by the keyword PFILE. The UNIQUE keyword is used to indicate that duplicate key values are not allowed for this file. The CUST field is identified by a K in position 17 of the last line as the key field for this record format.

The fields contained in this record format are listed in the order they are to appear in the record. The attributes for the fields are obtained from the physical file CUSMSTP. The physical file, in turn, refers to a field reference file to obtain the attributes for the fields. The field reference file is shown in Figure 8-2 on page 8-4.

Figure 8-1. Example of the Data Description Specifications for a Data Base File



IBM International Business Machines Corporation — DATA DESCRIPTION SPECIFICATIONS — GX21-7754- UM/050* Printed in U.S.A.

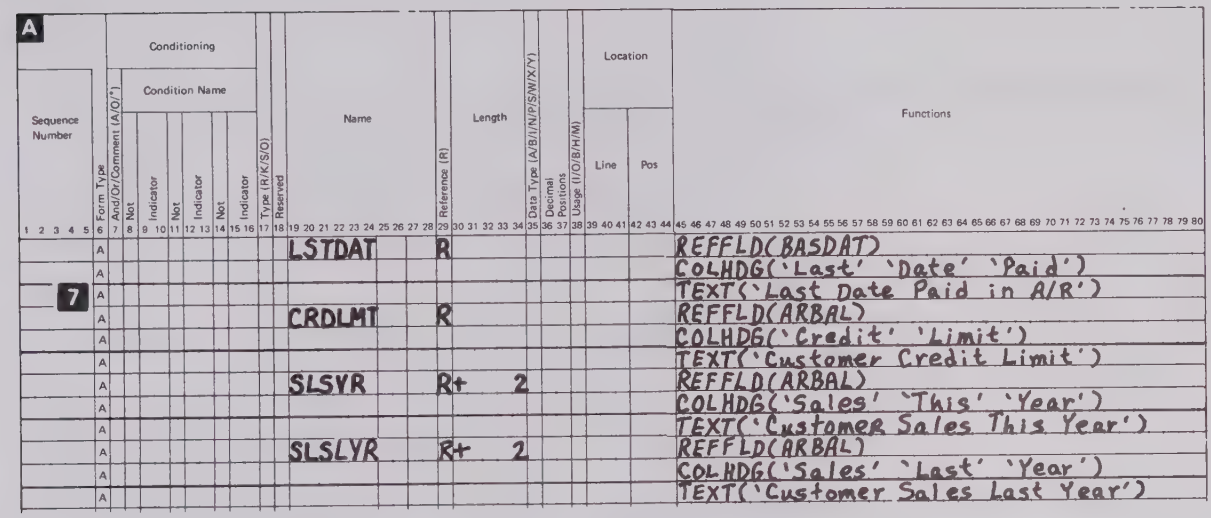
File | Keying Instruction | Graphic / Key | Description | Page of

Programmer | Date

	Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not	Indicator	Not	Indicator	Not	Indicator	Not	Indicator	Type (R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (A/B/I/N/P/S/W/X/Y)	Decimal Positions	Usage (I/O/B/H/M)	Location: Line	Pos	Functions
		A	**	FLDREF										DSTREF		DISTRIBUTION APPLICATION FIELD REFERENCE						
		A										R		DSTREF								TEXT('Distribution Field Ref')
		A	*	COMMON FIELDS USED AS REFERENCE																		
1		A												BASDAT		6	Ø					EDTCDE(Y)
		A																				TEXT('Base Date Field')
		A	*	FIELDS USED BY CUSTOMER MASTER FILE																		
2		A												CUST		5						CHECK(MF)
		A																				COLHDG('Customer' 'Number')
		A												NAME		2Ø						COLHDG('Customer Name')
3		A												ADDR	R							REFFLD(NAME)
		A																				COLHDG('Customer Address')
		A												CITY	R							REFFLD(NAME)
		A																				COLHDG('Customer City')
2		A												STATE		2						CHECK(MF)
		A																				COLHDG('State')
		A												SRHCOD		6						CHECK(MF)
		A																				COLHDG('Search' 'Code')
		A																				TEXT('Customer Number Search Code')
2		A												ZIP		5	Ø					CHECK(MF)
		A																				COLHDG('Zip' 'Code')
4		A												CUSTYP		1	Ø					RANGE(1 5)
		A																				COLHDG('Cust' 'Type')
		A																				TEXT('Customer Type 1=Gov 2=Sch 3=B+
		A																				us 4=Pvt 5=Oth')
5		A												ARBAL		8	2					COLHDG('Accts Rec' 'Balance')
6		A																				EDTCDE(J)
		A												ORDBAL	R							REFFLD(ARBAL)
		A																				COLHDG('A/R Amt in' 'Order File')
		A												LSTAMT	R							REFFLD(ARBAL)
		A																				COLHDG('Last' 'Amount' 'Paid')
		A																				TEXT('Last Amount Paid in A/R')
		A																				
		A																				
		A																				
		A																				
		A																				
		A																				
		A																				
		A																				
		A																				

*Number of sheets per pad may vary slightly.

Figure 8-2 (Part 1 of 2). Example of a Field Reference File



Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not	Indicator	Not	Indicator	Not	Indicator	Type (R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (A/B/I/N/P/S/W/X/Y)	Decimal Positions	Usage (I/O/B/H/M)	Location: Line	Pos	Functions
1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A																		REFFLD(BASDAT)
	A										LSTDAT	R							COLHDG('Last' 'Date' 'Paid')
7	A																		TEXT('Last Date Paid in A/R')
	A										CRDLMT	R							REFFLD(ARBAL)
	A																		COLHDG('Credit' 'Limit')
	A																		TEXT('Customer Credit Limit')
	A										SLSYR	R+	2						REFFLD(ARBAL)
	A																		COLHDG('Sales' 'This' 'Year')
	A																		TEXT('Customer Sales This Year')
	A										SLSLYR	R+	2						REFFLD(ARBAL)
	A																		COLHDG('Sales' 'Last' 'Year')
	A																		TEXT('Customer Sales Last Year')

Figure 8-2 (Part 2 of 2). Example of a Field Reference File

This example of a field reference file shows the definitions of the fields that are used by the CUSMSTL (customer master logical) file, which is shown in Figure 8-1 on page 8-3. The field reference file normally contains the definitions of fields that are used by other files. The following text describes some of the entries for this field reference file.

1 The BASDAT field is edited by the Y edit code, as indicated by the keyword EDTCDE (Y). If this field is used in an externally described output file for an RPG/400 program, the edit code used is the one specified in this field reference file; it cannot be overridden in the RPG/400 program. However, if the field is used in a program described output file for an RPG/400 program, an edit code must be specified for the field in the output specifications.

2 The CHECK(MF) entry specifies that the field is a mandatory fill field when it is entered from a display work station. Mandatory fill means that all characters for the field must be entered from the display work station.

3 The ADDR and CITY fields share the same attributes that are specified for the NAME field, as indicated by the REFFLD keyword.

4 The RANGE keyword, which is specified for the CUSTYP field, ensures that the only valid numbers that can be entered into this field from a display work station are 1 through 5.

5 The COLHDG keyword provides a column head for the field if it is used by the Interactive Data Base Utilities (IDU).

6 The ARBAL field is edited by the J edit code, as indicated by the keyword EDTCDE(J).

7 A text description (TEXT keyword) is provided for some fields. The TEXT keyword. is used for documentation purposes and appears in various listings.

Valid Keys for a Record or File

For a keyed sequence access path, you can define one or more fields in the data description specifications (DDS) to be used as the key fields for a record format. (These fields must not be floating-point fields.) All record types in a file do not have to have the same key fields. For example, an order header record can have the ORDER field defined as the key field, and the order detail records can have the ORDER and LINE fields defined as the key fields.

The key for a file is determined by the valid keys for the record types in that file. The file's key is determined in the following manner:

- If all record types in a file have the same number of key fields defined in DDS that are identical in attributes, the *key for the file* consists of all fields in the key for the record types. (The corresponding fields do not have to have the same name.) For example, if the file has three record types and the key for each record type consists of fields A, B, and C, then the file's key consists of fields A, B, and C. That is, the file's key is the same as the records' key.
- If all record types in the file do not have the same key fields, the key for the file consists of the key fields *common* to all record types. For example, a file has three record types and the key fields are defined as follows:

  - REC1 contains key field A.
  - REC2 contains key fields A and B.
  - REC3 contains key fields A, B, and C.

  Then the file's key is field A—the key field common to all record types.
- If no key field is common to all record types, there is no key for the file.

In RPG/400 you can specify a *search argument* on certain file operation codes to identify the record you want to process. RPG/400 compares the search argument with the key of the file or record, and processes the specified operation on the record whose key matches the search argument.

Valid Search Arguments

A search argument can be specified in the RPG/400 operations CHAIN, DELET, READE, SETGT, and SETLL that specify a file name or a record name.

For an operation to a file name, the maximum number of fields that you can specify in a search argument is equal to the total number of key fields valid for the file's key. For example, if all record types in a file do not contain all of the same key fields, a key list (KLIST) can be used to specify a search argument that is composed only of the number of fields common to all record types in the file. If a file contains three record types, then the key fields are defined as follows:

- REC1 contains key field A.
- REC2 contains key fields A and B.
- REC3 contains key fields A, B, and C.

The search argument can only be a single field with attributes identical to field A because field A is the only key field common to all record types.

For an operation to a record name, the maximum number of fields that you can specify in a search argument is equal to the total number of key fields valid for that record type.

If the search argument consists of one field, you can specify a literal, a field name, or a KLIST name with one KFLD. If the search argument is composed of more than one field (a composite key), you must specify a KLIST with multiple KFLDs.

The attributes of each field in the search argument must be identical to the attributes of the corresponding field in the file or record key. The attributes include the length, the data type (character or numeric), and the number of decimal positions. The attributes are listed in the key field information data table of the compiler listing. See example in Chapter 2, "Entering RPG/400 Specifications."

In all these file operations (CHAIN, DELET, READE, SETGT, and SETLL), a search argument can also be specified that contains fewer than the total number of fields valid for the file or record. Such a search argument refers to a *partial key*.

Referring to a Partial Key

The rules for the specification of a search argument that refers to a partial key are as follows:

- The search argument is composed of fields that correspond to the leftmost (high-order) fields of the key for the file or record.
- Only the rightmost fields can be omitted from the key list (KLIST) for a search argument that refers to a partial key. For example, if the total key for a file or record is composed of key fields A, B, and C, then the valid search arguments that refer to a partial key are field A, and fields A and B.
- Each field in the search argument must be identical in attributes to the corresponding key field in the file or record. The attributes include the length, data type (character or numeric), the number of decimal positions, and format (for example, packed or zoned).
- A search argument cannot refer to a portion of a key field.

If a search argument refers to a partial key, the file is positioned at the first record that satisfies the search argument or the record retrieved is the first record that satisfies the search argument. For example, the SETGT and SETLL operations position the file at the first record on the access path that satisfies the operation and the search argument. The CHAIN operation retrieves the first record on the access path that satisfies the search argument. The DELET operation deletes the first record on the access path that satisfies the search argument. The READE operation retrieves the next record if the portion of the key of that record (or the record of the specified type) on the access path matches the search argument. (For more information on these operation codes, see *Languages: RPG/400 Reference*.)

Processing Methods for Externally Described DISK Files

You can process externally described DISK files sequentially by key, randomly by key, randomly by relative record number, sequentially within limits, or consecutively (without a key or relative record number). A K in position 31 of the file description specifications for an externally described file indicates that the file is to be processed by key. If processing is sequential, records are retrieved in key sequence. If processing is random, key values are used to identify the records. A blank in position 31 indicates that the file is processed by relative record number, sequentially (arrival sequence), or randomly. Random or sequential processing is determined by the entries in positions 16 and 28 of the file description specifications and the operation code used on the calculation specifications (for example, CHAIN, SETLL, READ).

Program Described Disk Files

Program described files, which are identified by an F in position 19 of the file description specifications, can be described as indexed files, as sequential files, or as record address files.

Indexed Files

An indexed file is a program described DISK file whose access path is built on key values. You must create the access path for an indexed file by using data description specifications.

An indexed file is identified by an I in position 32 of the file description specifications.

The key fields identify the records in an indexed file. You specify the length of the key field in positions 29 and 30, the format of the key field in position 31, and the starting location of the key field in positions 35 through 38 of the file description specifications.

An indexed file can be processed sequentially by key, sequentially within limits, or randomly by key.

Valid Search Arguments

For a program described file, a search argument must be a single field and must be the same length as the key field that is defined on the file description specifications for the indexed file. The search argument cannot be a partial field.

The DDS specifies the fields to be used as a key field. Positions 35 through 38 of the file description specifications specify the starting position of the first key field. The entry in positions 29 and 30 of the file description specifications must specify the total length of the key as defined in DDS.

Figure 8-3 and Figure 8-4 show examples of how to use DDS to describe the access path for indexed files.

Sequential Files

Sequential files are files in which the order of the records in the file is based on the order in which the records are placed in the file (that is, in arrival sequence). For example, the tenth record placed in the file occupies the tenth record position.

Sequential files can be processed randomly by relative record number, consecutively, or by a record address file. In addition, the SETLL or SETGT operation code can be used to set limits on the file.

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754 UM/050* Printed in U.S.A.

File	Keying Instruction	Graphic	Description	Page of
Programmer	Date	Key		

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (1/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (B/A/P/S/B A/S/X/Y/N/I/M)	Decimal Positions	Usage (B/O/I/B/H/M)	Location: Line	Location: Pos	Functions
	A								R		FORMATA								PFILE(ORDDTLP)
	A																		Text('Access Path for Indexed File')
	A										FLDA		14						
	A										ORDER		5		Ø				
	A										FLDB		101						
	A								K		ORDER								
	A																		
	A																		

File Description Specifications

Line	Form Type	Filename	File Type: I/O/U/C/D	P/S/C/R/T/D/F	E	A/D	F/V/S/M/D/E	Block Length	Record Length	L/R	Mode of Processing: A/P/I/K	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Label: S/N/E/M	Name of Label Exit	Extent Exit for DAM	Storage Index	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8, UC	A/U	R/U/N
02	F	ORDDTLL	I	P			F		118			3	PI			15		DISK											
03	F																												
04	F																												

Continuation Lines: K | Option | Entry

Data description specifications must be used to create the access path for a program described indexed file.

In the DDS for the record format FORMATA for the logical file ORDDTLL, the field ORDER, which is 5 digits long, is defined as the key field, and is in packed format. The definition of ORDER as the key field establishes the keyed access for this file. Two other fields, FLDA and FLDB, describe the remaining positions in this record as character fields.

The program described input file ORDDTLL is described on the file description specifications as an indexed file. Positions 29 and 30 must specify the number of positions in the record required for the key field as defined in DDS, which is 3 positions. Positions 35 through 38 specify the starting position of the key field in the record, which is position 15. Because the file is defined as program described by the F in position 19, the RPG/400 compiler does not retrieve the external field-level description of the file at compilation time; therefore, the fields in the record must be described on the input specifications.

Figure 8-3. Using Data Description Specifications to Define the Access Path for an Indexed File

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754 UM/050* Printed in U.S.A.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name Indicator Not (N)	Conditioning: Condition Name Indicator Not (N)	Conditioning: Condition Name Indicator Not (N)	Name Type (B/R/K/S/O) Reserved	Name	Reference (R)	Length	Data Type (B A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (B/O/I/B/H/M)	Location: Line	Location: Pos	Functions
	A					R	FORMAT								PFILE(ORDDTLP)
	A														TEXT('Access Path for Indexed File')
	A						FLDA		14						
	A						ORDER		5						
	A						ITEM		5						
	A						FLDB		96						
	A					K	ORDER								
	A					K	ITEM								
	A														
	A														

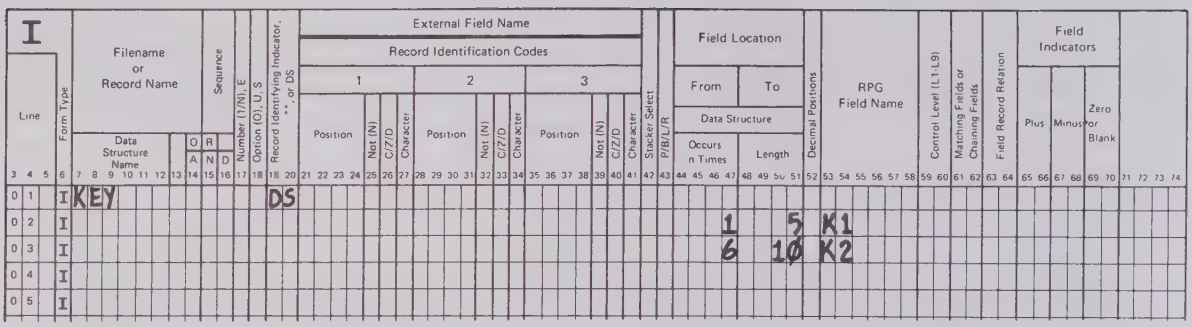
File Description Specifications

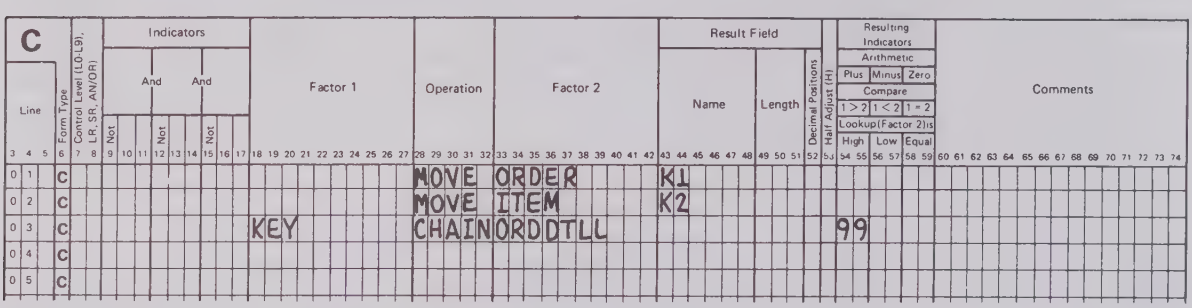
Line	Form Type	Filename	I/O/U/C/D	File Type: P/S/C/R/T/D/F	E	File Designation: A/D	File Format: F/V/S/M/D/E	Block Length	Record Length	L/R	Mode of Processing: A/P/I/K I/X/D/T/R/ or 2	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	External Record Name	Device	Symbolic Device	Labels S/N/E/M	Name of Label Exit	Extent Exit for DAM	Storage Index	Continuation Lines: K	Option	Entry	File Addition/Unordered A/U	Number of Tracks for Cylinder Overflow	Number of Extents	R/U/N	Tape Rewind	File Condition U1-UB, UC
02	F	ORDDTLL	I	P			F		120			10		A					DISK														
03	F																																
04	F																																

In this example, the data description specifications define two key fields for the record format FORMAT in the logical file ORDDTLL. For the two fields to be used as a composite key for a program described indexed file, the key fields must be contiguous in the record.

On the file description specifications, the length of the key field is defined as 10 in positions 29 and 30 (the combined number of positions required for the ORDER and ITEM fields). The starting position of the key field is described as 15 in positions 37 and 38. The starting position must specify the first position of the first key field.

Figure 8-4 (Part 1 of 2). Using Data Description Specifications to Define the Access Path (Composite Key) for an Indexed File





When the DDS specifies a composite key, you must build a search argument in the program to be used to CHAIN to the file. (A KLIST cannot be used for a program described file.) One way to do this is to create a data structure with subfields equal to the key fields defined in DDS. Then in the calculations, set the subfields equal to the value of the key fields, and use the data structure name as the search argument in the CHAIN operation.

In this example, the MOVE operations set the subfields K1 and K2 equal to the value of ORDER and ITEM, respectively. The data structure name (KEY) is then used as the search argument in the CHAIN operation.

Figure 8-4 (Part 2 of 2). Using Data Description Specifications to Define the Access Path (Composite Key) for an Indexed File

Record Address File

You can use a record address file to process another file. A record address file can contain (1) limits records that are used to process a file sequentially within limits, or (2) relative record numbers that are used to process a file by relative record numbers. The record address file itself must be processed sequentially.

A record address file is identified by an R in position 16 of the file description specifications. If the record address file contains relative record numbers, position 32 must contain a T. The name of the record address file must also be specified in positions 11 through 18 of the extension specifications, and the name of the file to be processed by the record address file must be specified in positions 19 through 26 of the extension specifications.

Limits Records

For sequential within limits processing, the record address file contains limits records. A limits record contains the lowest record key and the highest record key of the records in the file to be read.

The format of the limits records in the record address file is as follows:

- The low key begins in position 1 of the record; the high key immediately follows the low key, and no blanks can appear between the keys.
- Each record in the record address file can contain only one set of limits; therefore, the record length must be greater than or equal to twice the length of the record key.
- The low key and the high key in the limits record must be the same length. The length of the keys must be equal to the length of the key field of the file to be processed.
- A blank entry equal in length to the record key field causes RPG/400 to read the next record in the record address file.

Table 8-1. Processing Methods for DISK Files

Processing Method	Position 28	Position 31	Position 32
Externally Described Files			
With Keys			
Sequentially			
Randomly	Blank	K	Blank
Sequential within limits (by record address file)	Blank	K	Blank
Without Keys	L	K	Blank
Randomly/consecutively			
	Blank	Blank	Blank
Program Described Files			
With Keys (indexed file)			
Sequentially			
Randomly	Blank	A or P	I
Sequential within limits (by record address file)	Blank	A or P	I
Without Keys	L	A or P	I
Randomly/consecutively			
By record address file			Blank
As record address file (relative record numbers)	Blank	Blank	Blank
As record address limits file	Blank	Blank	T
	Blank	Blank	Blank
	Blank	A, P, or Blank	

Relative Record Numbers

For relative record number processing, the record address file contains relative record numbers. Each record retrieved from the file being processed is based on a relative record number in the record address file. A record address file containing relative record numbers cannot be used for limits processing. Each relative record number in the record address file is a 4-byte binary field; therefore, each 4 bytes of the record address file contains a relative record number. A minus 1 (-1 or hexadecimal FFFFFFFF) relative record number value stops the use of that relative record address file. End of file occurs when all records from the record address file have been processed.

Externally Described File as Program Described

A file that is externally described can be treated as a program described file in an RPG/400 program. To do this, specify an F in position 19 of the file description specifications, and describe the fields in the records on input and/or output specifications. When an F is specified in position 19 of the file description specifications for an externally described file, the compiler does not copy in the external description.

Methods for Processing Disk Files

Table 8-1 on page 8-13 shows the valid entries for positions 28, 31 and 32 of the file description specifications for the various file types and processing methods. The methods of disk file processing include:

- Relative record number processing
- Consecutive processing
- Sequential by key processing
- Random by key processing
- Sequential-within-limits processing.

Relative Record Number Processing

Random input or update processing by relative record number applies to full procedural files only. The desired record is accessed by the CHAIN operation code.

Relative record numbers identify the positions of the records relative to the beginning of the file. For example, the relative record numbers of the first, fifth, and seventh records are 1, 5, and 7, respectively.

For an externally described file, input or update processing by relative record number is determined by a blank in position 31 of the file description specifications and the use of the CHAIN operation code. Output processing by relative record number is determined by a blank in position 31 and the use of the RECNO option on a file description specifications continuation line for the file.

You can use the RECNO option for the file description specifications continuation line to specify a numeric field that contains the relative record number that specifies where a new record is to be added to this file. The RECNO field must be defined as numeric with zero decimal positions. The field length must be large enough to contain the largest record number for the file. A RECNO field must be specified if new records are to be placed in the file by using output specifications or a WRITE opera-

tion. When you update or add a record to a file by relative record number, the record must already have a place in the member. For an update, that place can be a valid existing record; for a new record, that place can be a deleted record. You can use the CL command INZPFM to initialize records for use by relative record number. The current relative record number is placed in the RECNO field for all retrieval operations or operations that reposition the file (for example, SETLL, CHAIN, READ).

Consecutive Processing

During consecutive processing, records are read in the order they appear in the file.

For output files and for input files that do not use random functions (such as SETLL, SETGT, CHAIN, or ADD), RPG/400 defaults to or operates as though SEQONLY(*YES) had been specified on the CL command OVRDBF (Override with Data Base File). (RPG/400 does not operate as though SEQONLY(*YES) had been specified for update files.) SEQONLY(*YES) allows multiple records to be placed in internal data management buffers; the records are then passed to RPG/400 one at a time on input. If, in the same job, two logical files use the same physical file, and one file is processed consecutively and one is processed for random update, a record could be updated that has already been placed in the buffer that is presented to the program. In this case, when the record is processed from the consecutive file, the record does not reflect the updated data. To prevent this problem, use the CL command OVRDBF and specify the option SEQONLY(*NO), which indicates that you do not want multiple records transferred for a consecutively processed file.

For more information on sequential only processing, see the *Programming: Data Base Guide*.

Sequential by Key Processing

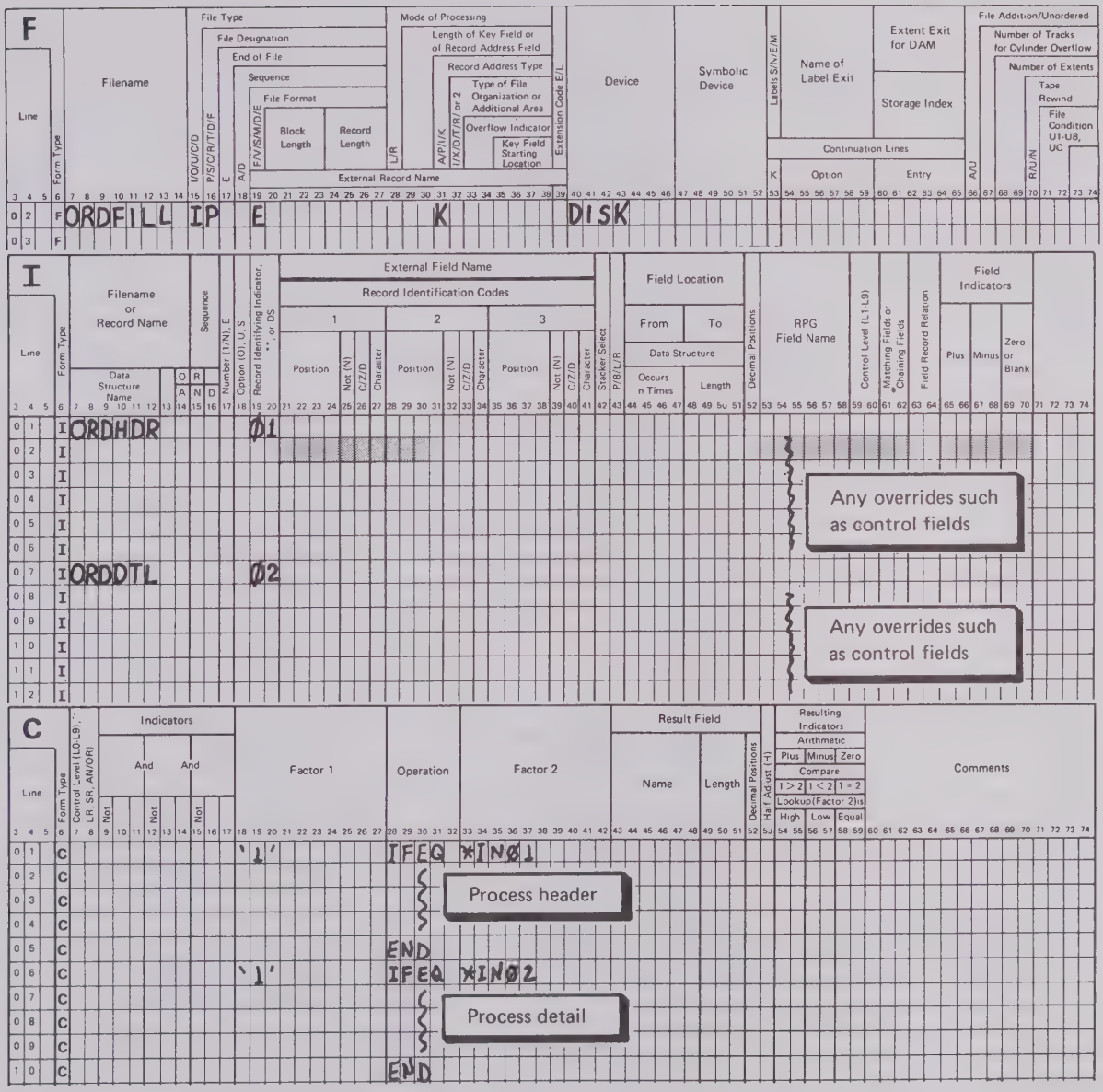
For the sequential-by-key method of processing, records are read from the file in key sequence.

The sequential-by-key method of processing is valid for keyed files used as primary, secondary, or full procedural files.

For output files and for input files that do not use random functions (such as SETLL, SETGT, CHAIN, or ADD) and that have only one record format, RPG/400 defaults to or operates as though SEQONLY(*YES) had been specified on the CL command OVRDBF. (RPG/400 does not operate as though SEQONLY(*YES) had been specified for update files.) SEQONLY(*YES) allows multiple records to be placed in internal data management buffers; the records are then passed to RPG/400 one at a time on input. If, in the same job, two files use the same physical file, and one file is processed sequentially and one is processed for random update, a record could be updated that has already been placed in the buffer that is presented to the program. In this case, when the record is processed from the sequential file, the record does not reflect the updated data. To prevent this problem, use the CL command OVRDBF and specify the option SEQONLY(*NO), which indicates that you do not want multiple records transferred for a sequentially processed file.

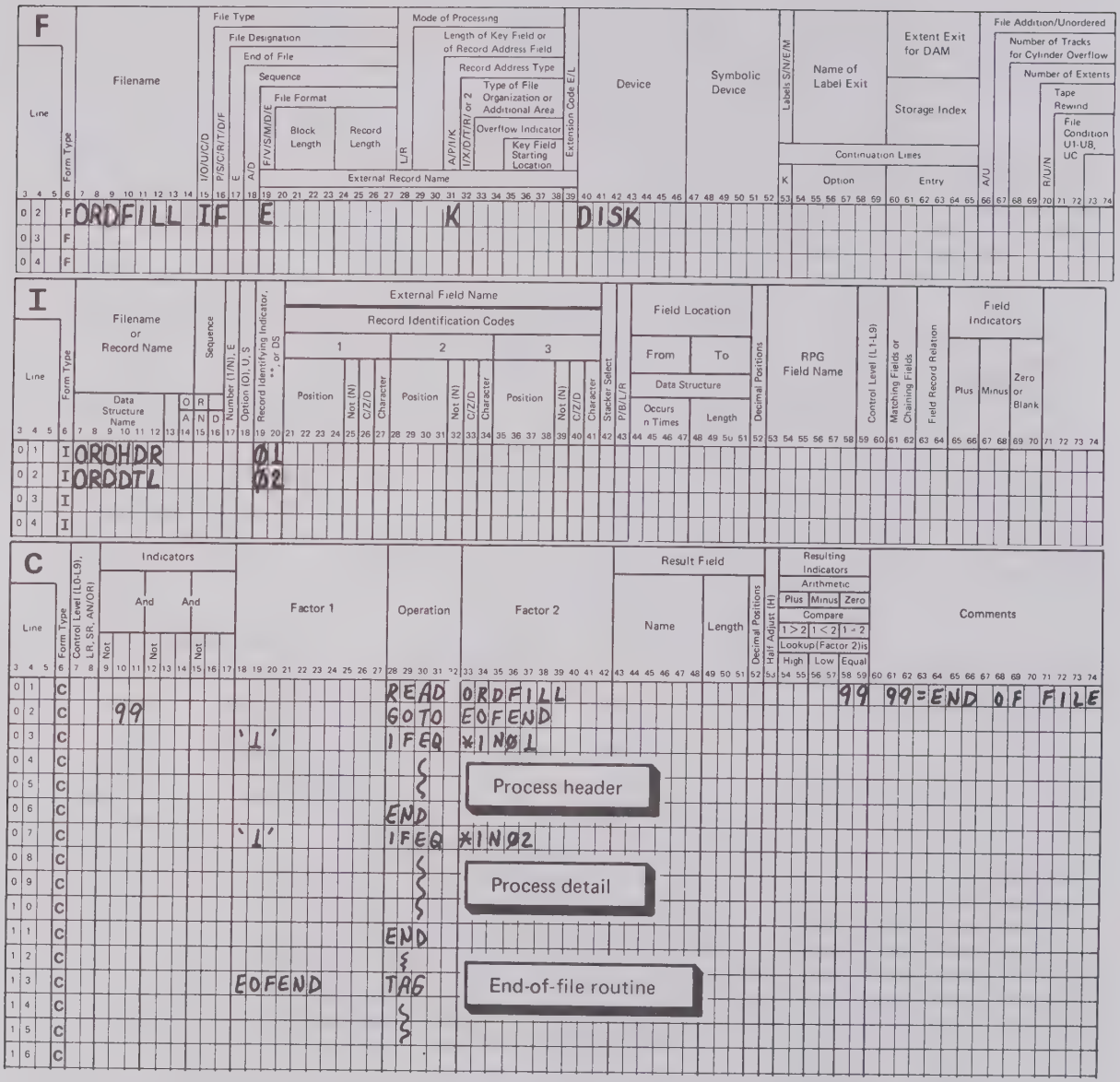
For more information on sequential only processing, see the *Programming: Data Base Guide*.

Figure 8-5 on page 8-17 shows different ways in which a header record and the detail records associated with the header record can be processed. Part 1 shows an example of the file being read sequentially by key; parts 2 through 4 show examples in which the READ operation code is used; part 5 shows the processing of these records by the matching record technique.



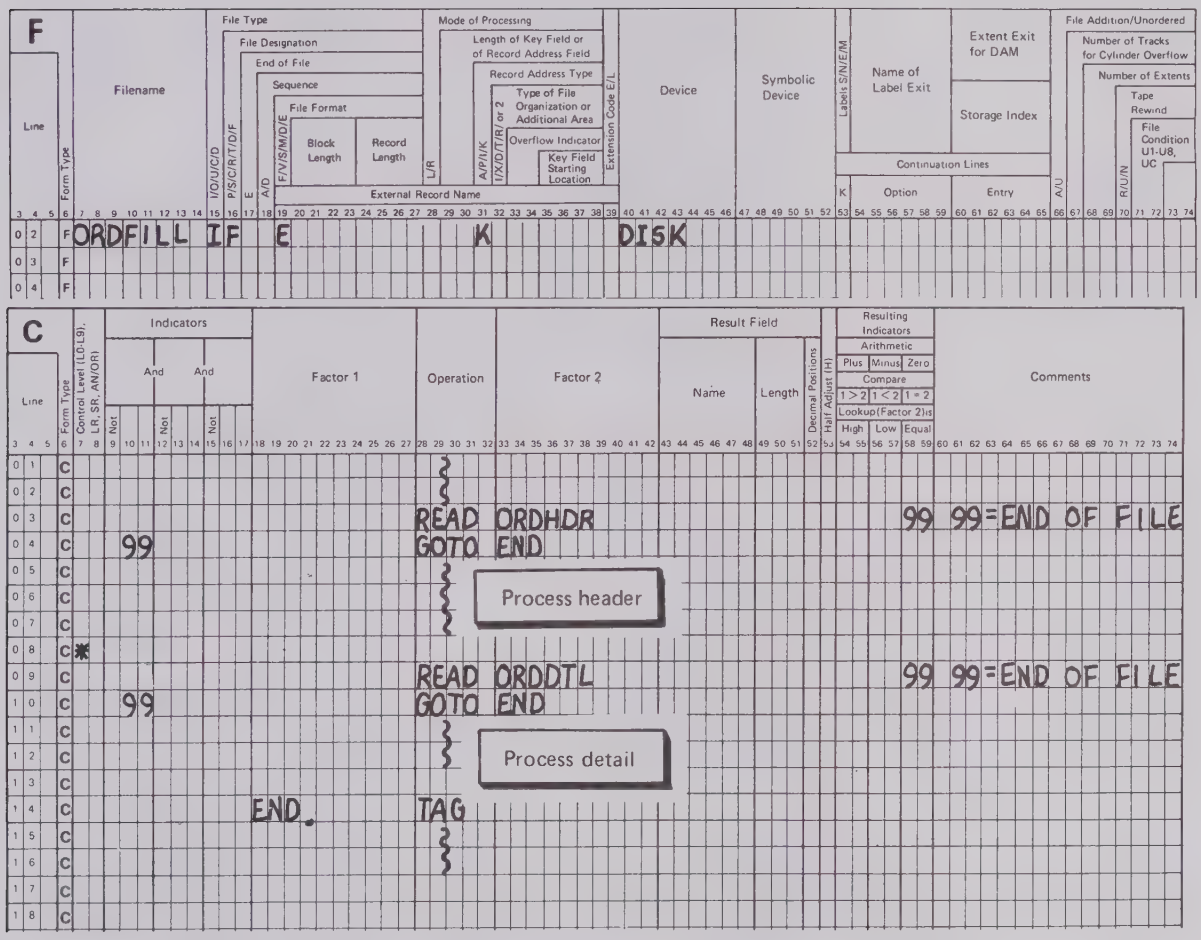
In this example, the order header record (ORDHDR) and the order detail record (ORDDTL) are contained in the same file (ORDFILL). The ORDFILL file is defined as a primary input file and is read sequentially by key. In the data description specifications for the file, the key for the ORDHDR record is defined as the ORDER field, and the key for the ORDDTL record is defined as the ORDER field plus the LINE (line number) field, which is a composite key. A record identifying indicator is assigned to each record; these record identifying indicators are used to control the processing for the different record types.

Figure 8-5 (Part 1 of 5). Processing Order Header and Order Detail Records



This example is the same as the previous example except that the ORDFILL file is defined as a full procedural file, and the reading of the file is done by the READ operation code. No control levels or match fields can be specified for a full procedural file. An end-of-file indicator must be specified on the READ operation. The two records (ORDHDR and ORDDTL) are contained in the same file, and a record identifying indicator is assigned to each record. The record identifying indicators are used to control the processing for the different record types.

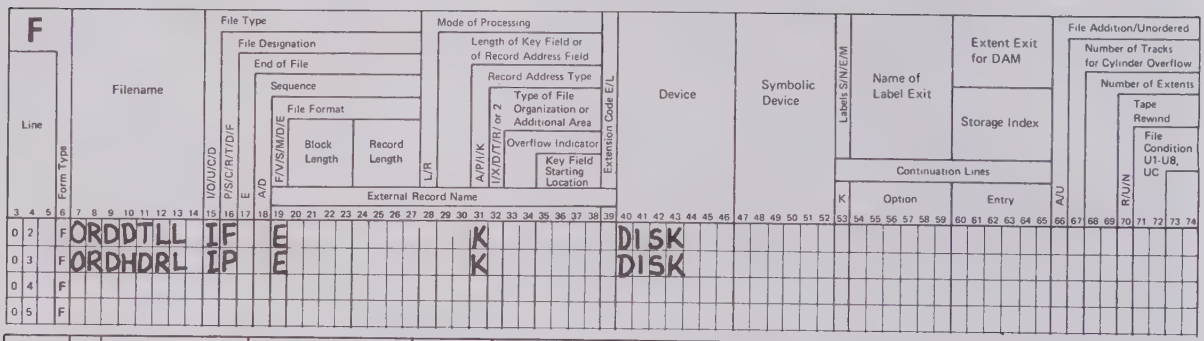
Figure 8-5 (Part 2 of 5). Processing Order Header and Order Detail Records

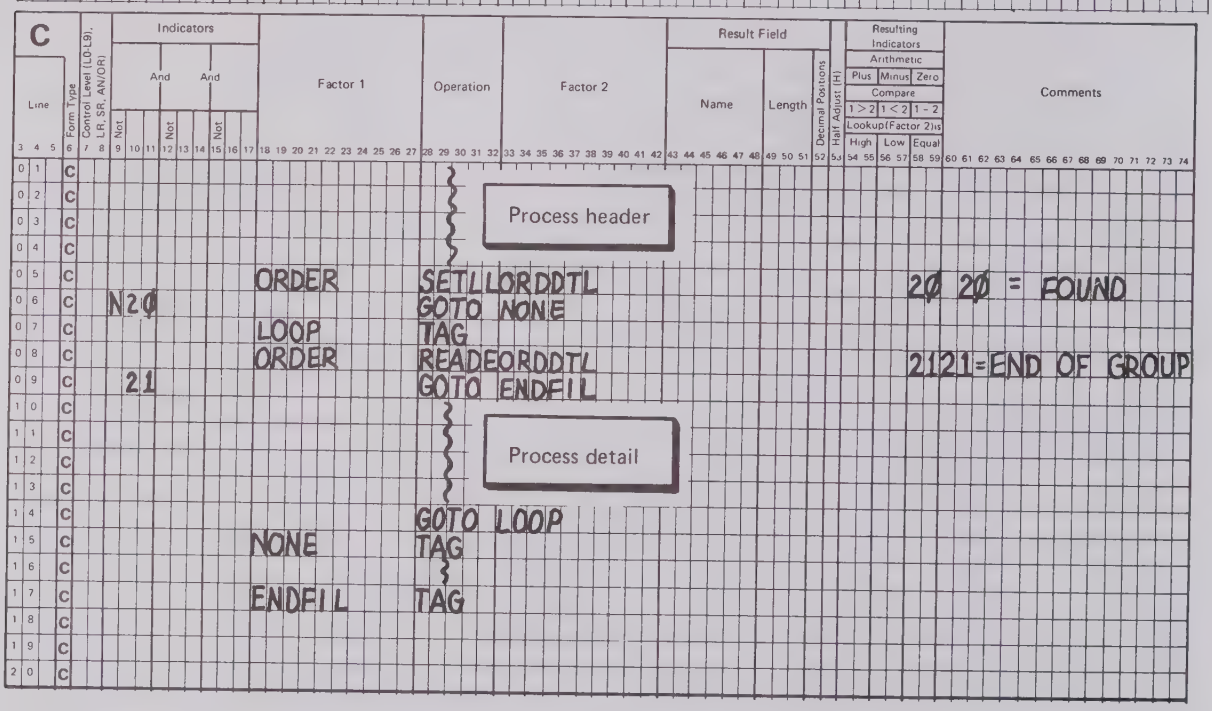


This example is similar to the one shown in Part 2 of this figure. However, the READ operation code is used to read each record (ORDHDR and ORDDTL) instead of reading the file. The program logic controls when each READ occurs. No record identifying indicators are needed because the program logic knows which record it is working with according to the record format name.

Figure 8-5 (Part 3 of 5). Processing Order Header and Order Detail Records

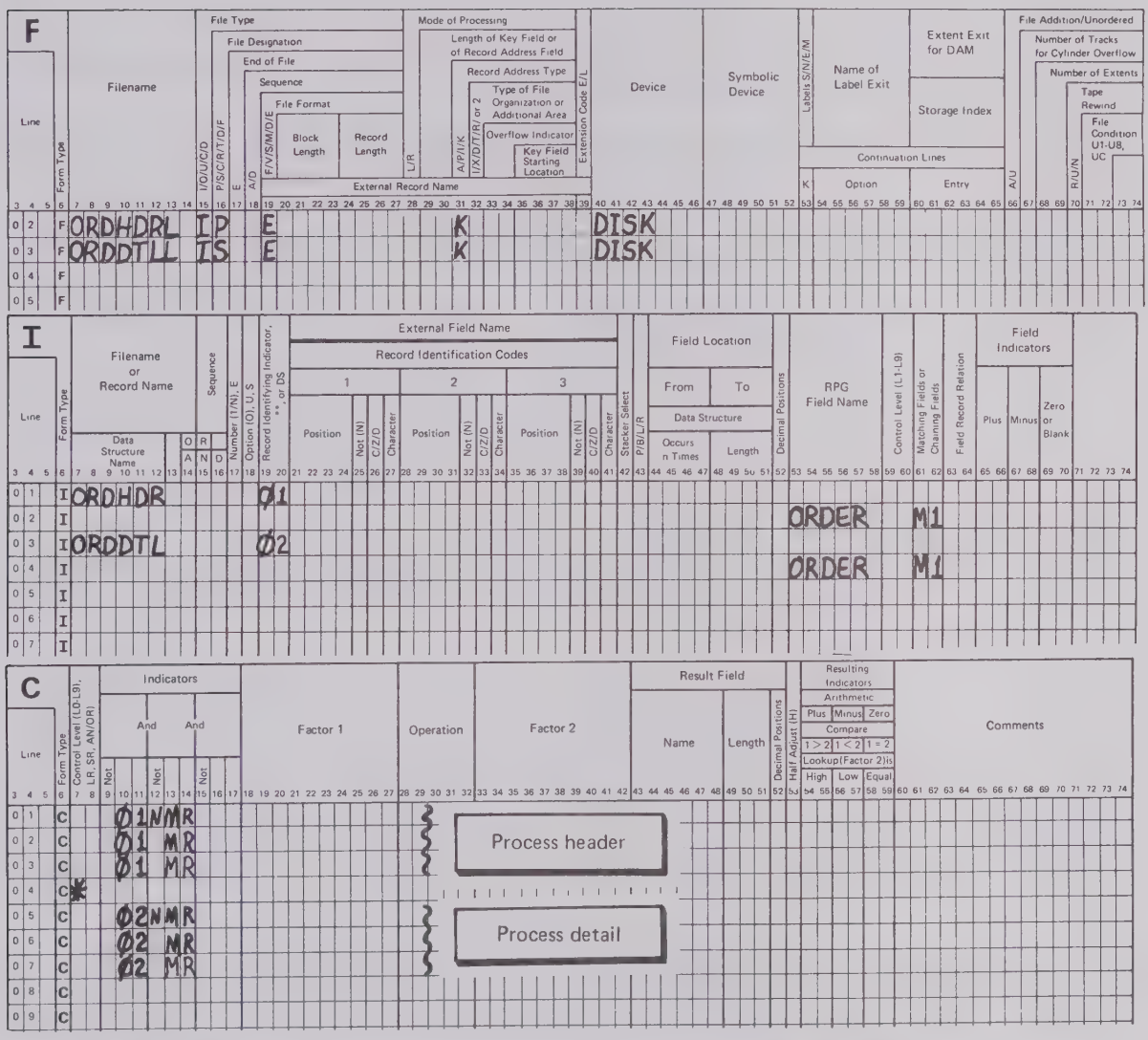
File Description Specifications





In this example, the order header records (ORDHDR) are contained in the ORDHDRL file, and the order detail records (ORDDTL) are contained in the ORDDTLL file. The ORDHDRL is defined as a primary input file, and the reading of records from the file is controlled by the program cycle. The ORDDTLL file is defined as a full procedural file, and the READE operation is used to read records from the file. The ORDER field in the SETLL operation is used to position the ORDDTLL file at the first ORDDTL record that has a key equal to or greater than the contents of the ORDER field. The ORDER field is used as the search argument for the READE operation. The READE operation retrieves the next ORDDTL record from the file if the key of the record is equal to the search argument specified in factor 1. If the key and search argument are not equal, the indicator specified in positions 58 and 59 is set on.

Figure 8-5 (Part 4 of 5). Processing Order Header and Order Detail Records



In this example, the order header records (ORDHDR) are contained in the ORDHDRL file, and the order detail records (ORDDTL) are contained in the ORDDTLL file. The ORDHDRL file is defined as a primary input file, and the ORDDTLL file is defined as a secondary input file. The order header and order detail records are processed as matching records, with the ORDER field in both records assigned the match level value of M1. Record identifying indicators 01 and 02 are assigned to the records to control the processing for the different record types.

Figure 8-5 (Part 5 of 5). Processing Order Header and Order Detail Records

Random by Key Processing

For the random-by-key method of processing, you specify a search argument that identifies the key of the record to be read in factor 1 of the calculation specifications for the CHAIN operation. Figure 8-6 on page 8-23 shows an example of an externally described DISK file being processed randomly by key. The specified record can be read from the file either during detail calculations or during total calculations.

The random-by-key method of processing is valid for a full procedural file designated as an input file or an update file.

For an externally described file, position 31 of the file description specifications must contain K, which indicates that the file is processed according to an access path that is built on keys The data description specifications (DDS) for the file specifies the field that contains the key value (the key field). Position 32 of the file description specifications must be blank.

You must designate a program described file as an indexed file (I in position 32), and position 31 of the file description specifications must contain an A or a P. The length of the key field is identified in positions 29 and 30 of the file description specifications, and the starting location of the key field is identified in positions 35 through 38. Data description specifications must be used to create the access path for a program described input file (see "Indexed Files" on page 8-8).

Sequential within Limits Processing

Sequential-within-limits processing by a record address file is specified by an L in position 28 of the file description specifications and is valid for a file with a keyed access.

You can specify sequential-within-limits processing for an input or an update file that is designated as a primary, secondary, or full procedural file. The file can be externally described or program described (indexed). For an externally described file, the keys should be in ascending sequence.

To process a file sequentially within limits from a record address file, the program reads:

- A limits record from the record address file.
- Records from the file being processed within limits with keys greater than or equal to the low record key and less than or equal to the high record key in the limits record. If the two limits supplied by the record address file are equal, only the records with the specified key are retrieved.

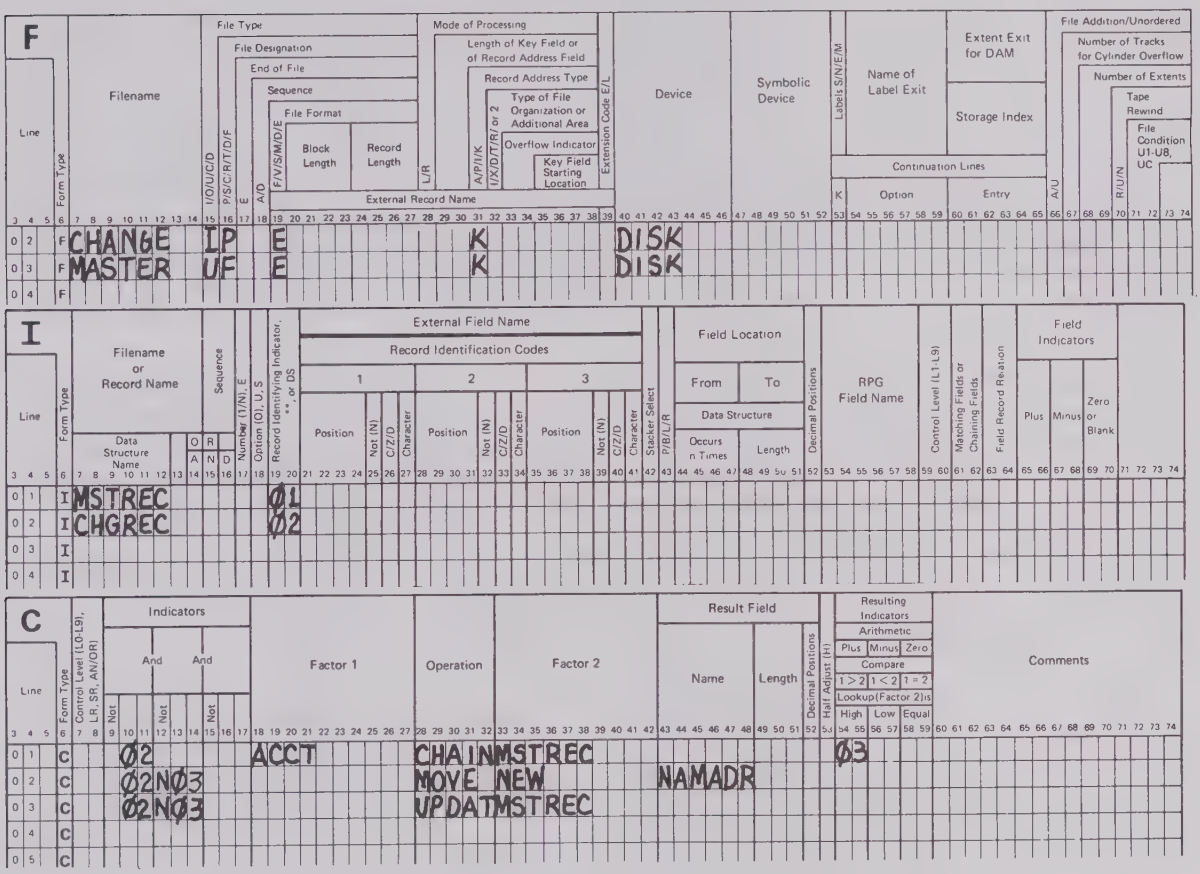
The program repeats this procedure until the end of the record address file is reached.

Figure 8-7 on page 8-24 shows an example of an indexed file being processed sequentially within limits. Figure 8-8 on page 8-26 shows the same example with externally described files instead of program described files.

Keyed Processing Examples

Figure 8-9 on page 8-27 shows an example of processing certain records in a group. Figure 8-10 on page 8-29 shows examples of how to process the first record in a file and the last record in a file.

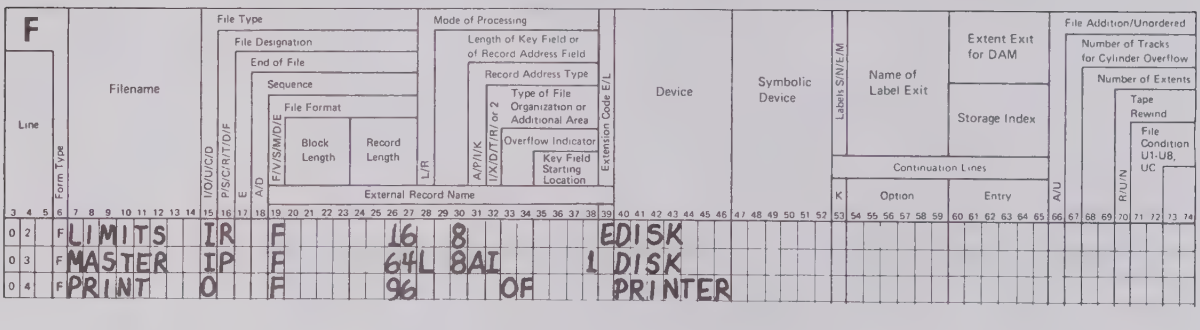
File Description Specifications



The update file MASTER is to be processed by keys. The data description specifications (DDS) for each of the externally described files (MASTER and CHANGE) identify the ACCT field as the key field. As each record is read from the primary input file, CHANGE, the account number field (ACCT) is used as the search argument to chain to the corresponding record in the MASTER file. Input specifications are used to assign record identifying indicators to the records in the CHANGE and MASTER files. The MASTER file contains one record format MSTREC that contains two fields, ACCT and NAMADR (name and address). The CHANGE file contains one record format CHGREC that contains two fields, ACCT and NEW. The data in the NEW field must be moved into the NAMADR field before the MSTREC can be updated.

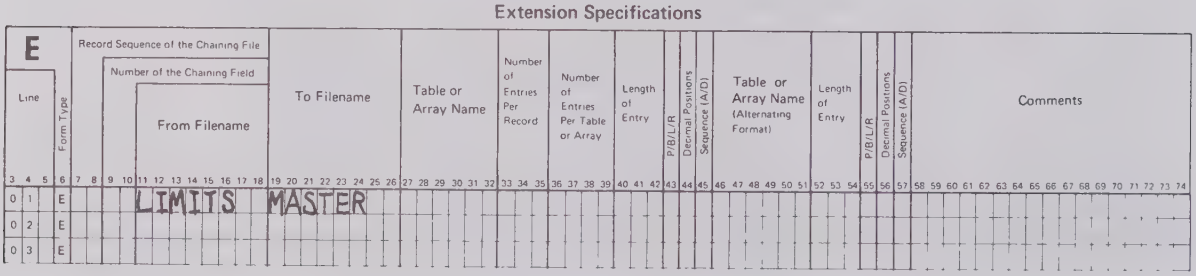
Figure 8-6. Random Processing of an Externally Described DISK File by Key

File Description Specifications



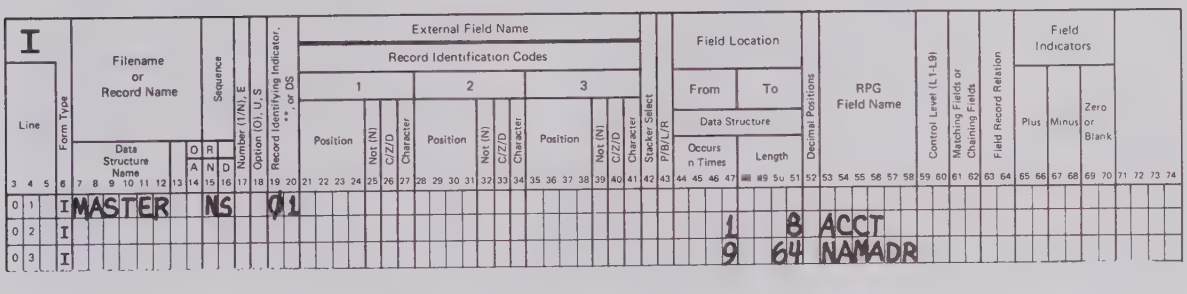
The input file MASTER, which is a program described file (F in position 19), is described as an indexed file to be processed by keys. (The access path for an indexed file must be created by data description specifications.) MASTER is processed sequentially within limits (L in position 28) by the record address file LIMITS. Each set of limits from the record address file consists of the low and high account numbers of the records in the MASTER file to be processed. Because the account number key field (ACCT) is eight positions long, each set of limits consists of two 8-position keys.

Extension Specifications

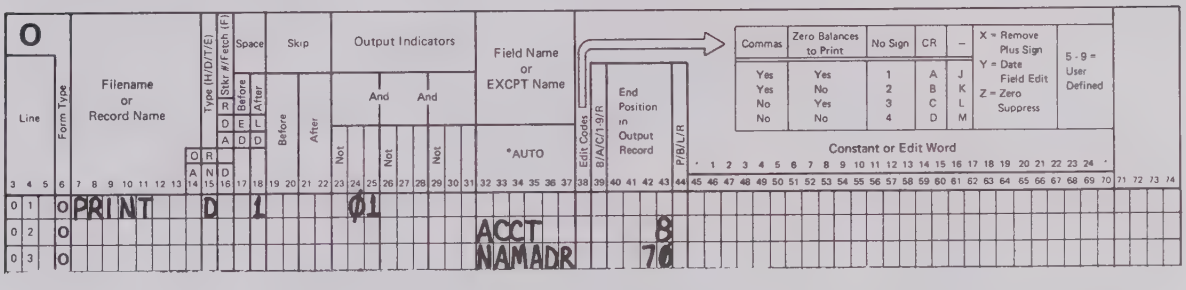


The record address file name LIMITS must be specified in positions 11 through 18 of the extension specifications. The name of the file to be processed by the record address file must be specified in positions 19 through 26 of the extension specifications.

Figure 8-7 (Part 1 of 2). Processing an Indexed File Sequentially within Limits



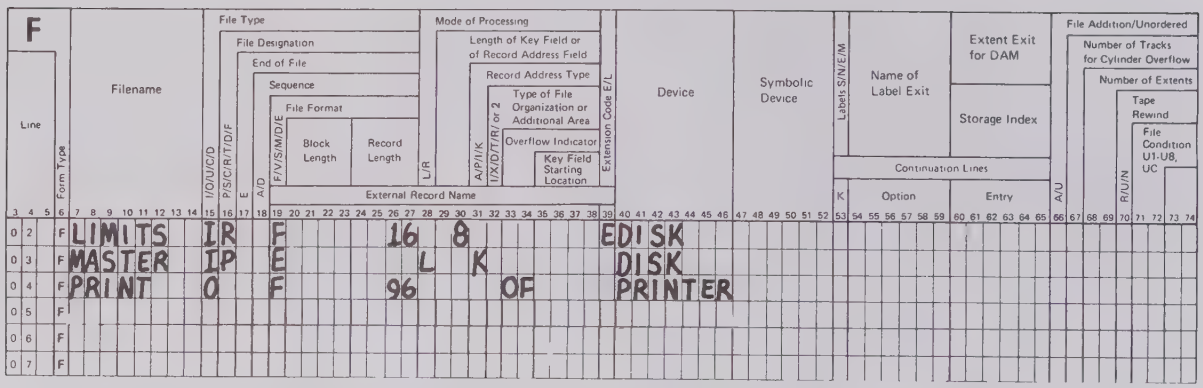
Input specifications must be used to describe the records in the program described file MASTER.

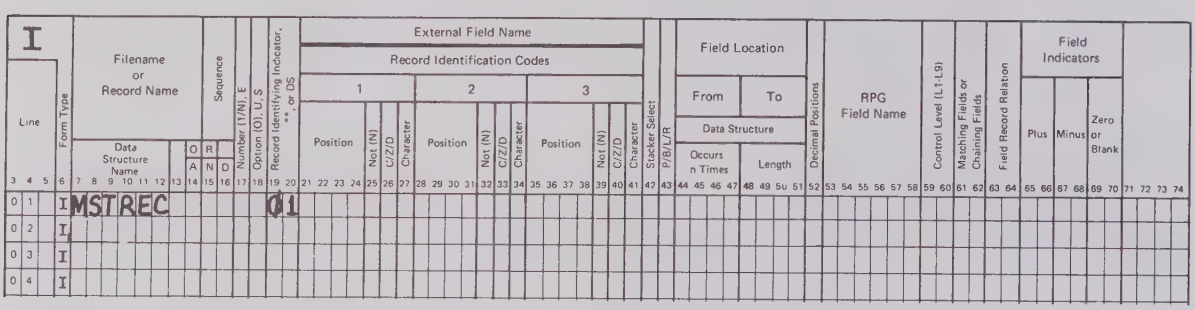


As MASTER is processed within each set of limits, the corresponding records are printed. Processing of the MASTER file is complete when the record address file LIMITS reaches end of file.

Figure 8-7 (Part 2 of 2). Processing an Indexed File Sequentially within Limits

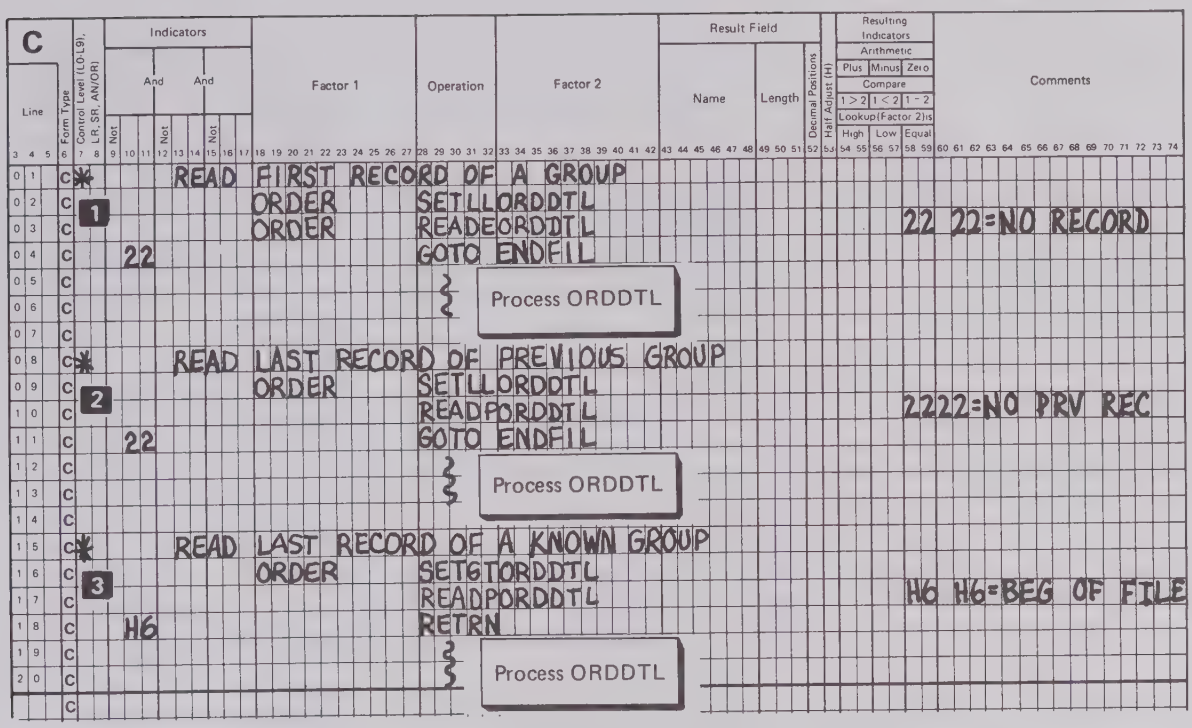
File Description Specifications





If the program shown in Figure 8-7 on page 8-24 used externally described files, the file description specifications would be coded as shown above. The input specifications are used to assign a record identifying indicator to the record in the externally described file. The MASTER file contains the record format MSTREC. The external descriptions for the file identify the key fields. These keys should be in ascending sequence.

Figure 8-8. Processing an Externally Described File Sequentially within Limits



1 This example shows how to retrieve the first record of a group. The SETLL operation is used to position the file at the first ORDDTL record that has a key equal to or greater than the search argument contained in the ORDER field. The READE operation reads the next ORDDTL record from the file if the key of the record is equal to the search argument (the ORDER field) specified in factor 1. If the key is not equal to the search argument, the indicator specified in positions 58 and 59 is set on.

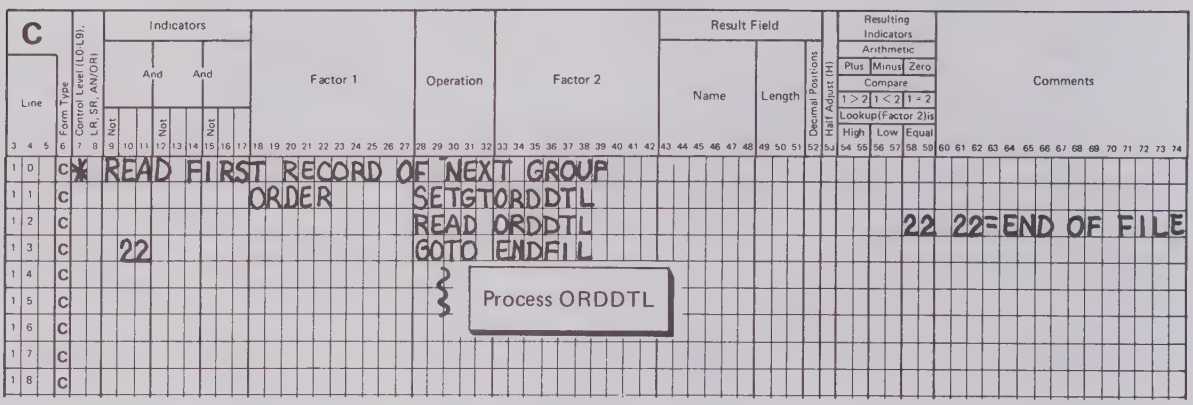
2 This example shows how to retrieve the last record of the previous group. The ORDER field, which contains the key of the current group, is used in the SETLL operation to position the file at the first ORDDTL record that has a key equal to or greater than the search argument contained in the ORDER field. The READP operation then reads the prior record. If there is no prior record in the file, the program branches to the ENDFIL routine.

3 This example retrieves the last record of a group. One or more records for the group exist in the file. The SETGT operation positions the file at the next record that contains a key value greater than the search argument contained in the ORDER field. For example, if the ORDER field contains a value of 10, SETGT positions the file at the record that contains a key value greater than 10:

```
          Keys
          9
          9
          10
          10
SETGT---> 11
```

The READP operation then reads the prior record of the ORDDTL record format, therefore reading the last record of the previous group. READP requires an end-of-file indicator in positions 58 and 59; therefore, if the beginning of the file is encountered, the halt indicator H6 is set on and the program abnormally ends.

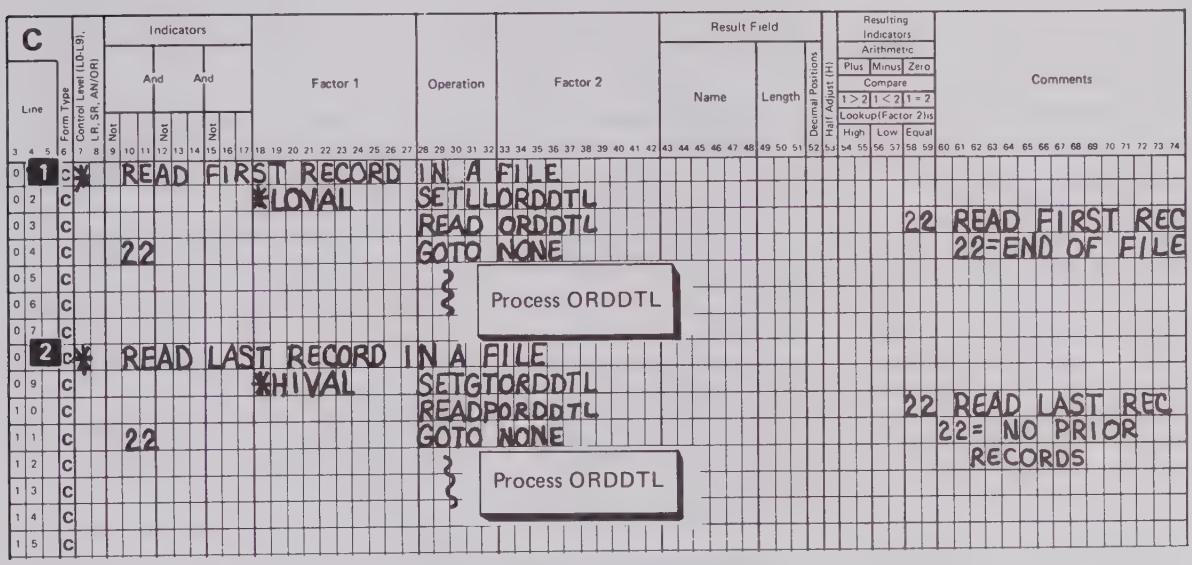
Figure 8-9 (Part 1 of 2). Processing Certain Records in a Group



4 Reading the first record of the next group requires the SETGT operation to position the file and the READ operation. The ORDER field, which contains the key of the current group, is specified in factor 1 of the SETGT operation. The READ operation is then used to read the first record of the next group. An indicator must be specified in positions 58 and 59 of the READ operation to test for end of file. This technique can be used if the program knows the key value for a group of records or for a specific record and wants the next group. SETGT can be used to eliminate reading unwanted records that would be bypassed.

Note: All the file operation codes in these examples specify record format names rather than file names because the files are externally described.

Figure 8-9 (Part 2 of 2). Processing Certain Records in a Group



1 After the file is opened, the first record is retrieved by a subsequent READ operation. To access the first record in a file after some processing has been done, the figurative constant *LOVAL (assuming ascending key sequence) is used in a SETLL operation to set the lower limits. The READ operation reads the next record of the ORDDTL record format. If no records exist, end of file occurs, and the program branches to the NONE routine.

2 To access the last record in a file, the figurative constant *HIVAL (assuming ascending key sequence) is used in a SETGT operation to position the file at the next record with a key field value greater than the value specified in factor 1. The READP operation reads the prior record, which in this example is the last record in the file.

Figure 8-10. Processing Certain Records in a File

Valid File Operations

Figure 8-11 on page 8-30 and Figure 8-12 on page 8-31 show the valid file operation codes allowed for DISK files processed by keys and for DISK files processed by nonkeyed methods, respectively. The operations shown in these figures are valid for both externally described DISK files and program described DISK files.

Before running your program, you may override a file to another file. In particular, you may override a sequential file in your program to an externally described, keyed file. (The file is processed as a sequential file.) You may also override a keyed file in your program to another keyed file, providing the key fields are compatible (for example, the overriding file must not have a shorter key field than you specified in your program).

Figure 8-13 on page 8-32 is a series of processing charts for DISK files.

File Description Specifications Positions					Calculation Specifications Positions
15	16	28[1]	31[2]	66	28-32
I	P/S		K/A/P		CLOSE, FEOD
I	P/S		K/A/P	A	WRITE, CLOSE, FEOD
I	P/S	L	K/A/P		CLOSE, FEOD
U	P/S		K/A/P		UPDAT, WRITE, CLOSE, FEOD
U	P/S		K/A/P	A	UPDAT, DELET, WRITE, CLOSE, FEOD
U	P/S	L	K/A/P		UPDAT, CLOSE, FEOD
I	F		K/A/P		READ, READE, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
I	F		K/A/P	A	WRITE, READ, READE, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
I	F	L	K/A/P		READ, OPEN, CLOSE, FEOD
U	F		K/A/P		READ, READE, READP, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
U	F		K/A/P	A	WRITE, UPDAT, DELET, READ, READE, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
U	F	L	K/A/P		READ, UPDAT, OPEN, CLOSE, FEOD
O	Blank		K/A/P	A	WRITE (add new records to a file), OPEN, CLOSE, FEOD
O	Blank		K/A/P		WRITE (initial load of a new file)[3], OPEN, CLOSE, FEOD

[1]An L must be specified in position 28 to specify sequential-within-limits processing by a record address file for an input or an update file.
[2]Externally described files require a K in position 32; program described files require an A or P in position 31 and an I in position 32.
[3]An A in position 66 is not required for the initial loading of records into a new file; however, if A is specified in position 66 then ADD must be specified on the output specifications. The file must have been created with the OS/400 CREATE FILE command.

Figure 8-11. Valid File Operations for Keyed Processing Methods (Random by Key, Sequential by Key, Sequential within Limits)

File Description Specifications Positions					Calculation Specifications Positions
15	16	31	54-59	66	28-32
I	P/S	Blank			CLOSE, FEOD
I	P/S	Blank	RECNO		CLOSE, FEOD
U	P/S	Blank			UPDAT, DELET, CLOSE, FEOD
U	P/S	Blank	RECNO		UPDAT, DELET, CLOSE, FEOD
I	F	Blank			READ, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
I	F	Blank	RECNO		READ, READP, SETLL, SETGT,
U	F	Blank			READ, READP, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
U	F	Blank	RECNO		READ, READP, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
I	R	A/P/Blank[3]			OPEN, CLOSE, FEOD
I	R	Blank[4]			OPEN, CLOSE, FEOD
O	Blank	Blank	RECNO	A	WRITE[1] (add records to a file), OPEN, CLOSE, FEOD
O	Blank	Blank	RECNO		WRITE[2] (initial load of a new file), OPEN, CLOSE, FEOD
O	Blank	Blank	Blank		WRITE (sequentially load or extend a file), OPEN, CLOSE, FEOD

[1]The RECNO field that contains the relative record number must be set prior to the WRITE operation or if ADD is specified on the output specifications.
[2]An A in position 66 is not required for the initial loading of the records into a new file; however, if A is specified in position 66, then ADD must be specified on output specifications. The file must have been created with the OS/400 CREATE FILE command.
[3]If position 31 is blank for a record address limits file, the format of the keys in the record address file is the same as the format of the keys in the file being processed.
[4]A record address file containing relative record numbers requires a T in position 32.

Figure 8-12. Valid File Operations for Nonkeyed Processing Methods (Sequential, Random by Relative Record Number, and Consecutive)

Valid File Operations

Notes:

1. Shaded positions must be blank, and positions without entries are program dependent.
2. The RECNO option must be specified in positions 54 through 59 for an output file that uses relative record numbers. The option can be specified for input/update files. RPG places the relative record number of any retrieved record in the RECNO field.

Figure 8-13 (Part 1 of 4). Processing Chart for Externally Described DISK Files (Processing without Keys)

Note: Shaded positions must be blank, and positions without entries are program dependent.

Figure 8-13 (Part 2 of 4). Processing Chart for Externally Described DISK Files (Processing without Keys)

Valid File Operations

File Description Specifications

	Valid File Operations
In Sequence by Relative Record Number	
No ADD	CLOSE, FEOD
No ADD	
No ADD	UPDAT, DELET, CLOSE, FEOD
No ADD	
Random/Sequential	
No ADD	READ, READP, SETLL, SETGT, CHAIN, OPEN, CLOSE, FEOD
No ADD	READ, READP, SETLL, SETGT, CHAIN, UPDAT, DELET, OPEN, CLOSE, FEOD
By Record Address File	
	CLOSE, FEOD
	READ, OPEN, CLOSE, FEOD
	UPDAT, CLOSE, FEOD
	READ, UPDAT, OPEN, CLOSE, FEOD
As Record Address File	
Limits records	
Relative record numbers	OPEN, CLOSE, FEOD
Table	
Output without Keys	
With ADD or WRITE	WRITE (add records), OPEN, CLOSE, FEOD WRITE (initial load), OPEN, CLOSE, FEOD WRITE (initial load or extend), OPEN, CLOSE, FEOD

Notes:

1. Shaded positions must be blank, and positions without entries are program dependent.
2. The RECNO option must be specified in positions 54 through 59 for an output file that uses relative record numbers. The option can be specified for input/update files. RPG places the relative record number of any retrieved record in the RECNO field.
3. All WRITE and UPDAT operations to a program described file require a data structure name in the result field.

Figure 8-13 (Part 3 of 4). Processing Chart for Externally Described DISK Files (Processing without Keys)

Notes:

1. Shaded positions must be blank, and positions without entries are program dependent.
2. The entry in position 32 (A or P) depends on the format of the keys in the file.
3. All WRITE and UPDAT operations to a program described file require a data structure name in the result field.

Figure 8-13 (Part 4 of 4). Processing Chart for Externally Described DISK Files (Processing without Keys)

Chapter 9. Using WORKSTN files

The WORKSTN file allows an RPG/400 program to communicate interactively with a work station user or to use the Intersystem Communications Function (OS/400-ICF), hereafter referred to as ICF, to communicate with other programs. This chapter describes:

- Intersystem Communications Function (ICF)
- Externally described WORKSTN files
- Program described WORKSTN files
- Multiple device files.

The chapter also includes a number of examples for using WORKSTN files.

Intersystem Communications Function (ICF)

You can use the Intersystem Communications Function (ICF) to write programs that will communicate with (send data to and receive data from) other application programs on other systems.

To use Intersystem Communications Function, define a WORKSTN file in your program that refers to an ICF device file. Use either the system supplied file QICDMF or a file created using the OS/400 command CRTICFF.

You code for ICF by using the ICF as a file in your program similar to display files. This refers to the file that contains the communications formats required for the sending and receiving of data between systems.

For further information on ICF, refer *Communications: Programmer's Guide.*

Externally Described WORKSTN Files

An RPG/400 WORKSTN file may use an externally described display device file, which contains file information and a description of the fields in the records to be written.

In addition to the field descriptions (such as field names and attributes), the data description specifications (DDS) for a display device file are used to:

- Format the placement of the record on the screen by specifying the line number and position number entries for each field and constant.
- Specify attention functions such as underlining and highlighting fields, reverse image, or a blinking cursor.
- Specify validity checking for data entered at the display work station. Validity checking functions include detecting fields where data is required, detecting mandatory fill fields, detecting incorrect data types, detecting data for a specific range, checking data for a valid entry, and processing modulus 10 or 11 check digit verification.

- Control screen management functions such as if fields are to be erased, overlaid, or retained when new data is displayed.
- Associate indicators 01 through 99 with command attention keys or command function keys. If a function key is described as a command function key (CF), both the response indicator and the data record (with any modifications entered on the screen) are returned to the program. If a function key is described as a command attention key (CA), the response indicator is returned to the program but the data record remains unmodified. Therefore, input-only character fields are blank and input-only numeric field are filled with zeros, unless these fields have been initialized otherwise.
- Assign an edit code (EDTCDE) or edit word (EDTWRD) keyword to a field to specify how the field's values are to be displayed.
- Specify subfiles.

A display device record format contains three types of fields:

- *Input fields.* Input fields are passed from the device to the program when the program reads a record. Input fields can be initialized with a default value; if the default value is not changed, the default value is passed to the program. Input fields that are not initialized are displayed as blanks into which the work station user can enter data.
- *Output fields.* Output fields are passed from the program to the device when the program writes a record to a display. Output fields can be provided by the program or by the record format in the device file.
- *Output/input (both) fields.* An output/input field is an output field that can be changed and, therefore, become an input field if it is changed. Output/input fields are passed from the program when the program writes a record to a display and passed to the program when the program reads a record from the display. Output/input fields are used when the user is to change or update the data that is written to the display from the program.

If you specify the keyword INDARA in the DDS for a WORKSTN file, RPG/400 passes indicators to the WORKSTN file in a separate indicator area, and not in the input/output buffer.

For a detailed description of an externally described data device file and for a list of valid DDS keywords, see the *Programming: Data Description Specifications Reference.*

Figure 9-1 on page 9-3 shows an example of the DDS for a display device file.

Processing an Externally Described WORKSTN File

When an externally described WORKSTN file is processed, OS/400 transforms data from the program to the format specified for the file and displays the data. When data is passed to the program, the data is transformed to the format used by the program.

OS/400 provides device control information for processing input/output operations for the device. When an input record is requested from the device, OS/400 issues the request, then removes device control information from the data before passing

the data to the program. In addition, OS/400 can pass indicators to the program indicating which fields, or if any fields, in the record have been changed.

When the program requests an output operation, it passes the output record to OS/400. OS/400 provides the necessary device control information to display the record. OS/400 also adds any constant information specified for the record format when the record is displayed.

IBM International Business Machines Corporation — **DATA DESCRIPTION SPECIFICATIONS** — GX21-7754 UM/050* Printed in U.S.A.

File | Programmer | Date | Keying Instruction | Graphic Key | Description | Page of

Seq Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (M/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (M A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (M O/I/B/H/M)	Location: Line	Location: Pos	Functions
	A	**		ITEM MASTER INQUIRY															
	A																		REF(DSTREF) 1
	A								R		PROMPT								TEXT('Item Prompt Format')
	A			73N61															OVERLAY
	A																	2	CA01(98 'End of Program') 3
	A																1	2	'Item Inquiry'
	A																3	2	'Item Number'
	A										ITEM	R				I	3	15	PUTRETAIN 4
	A		61																ERRMSG('Invalid Item Number' 61) 5
	A								R		RESPONSE								TEXT('Response Format')
	A																	2	OVERLAY
	A																		LOCK 6
	A																5	2	'Description'
	A										DESCRP	R					5	15	
	A																5	37	'Price'
	A										PRICE	R					5	44	
	A																7	2	'Warehouse Location' 7
	A										WHSLOC	R					7	22	
	A																9	2	'On Hand'
	A										ONHAND	R					9	10	
	A															8	9	19	'Allocated'
	A										ALLOC	R					9	30	
	A																9	40	'Available'
	A										AVAIL	R					9	51	
	A																		
	A																		
	A																		

Figure 9-1 (Part 1 of 2). Example of the Data Description Specifications for a Display Device File

This display device file contains two record formats: PROMPT and RESPONSE.

1. The attributes for the fields in this file are defined in the DSTREF field reference file.
2. The OVERLAY keyword is used so that both record formats can be used on the same display.
3. Function key 1 is associated with indicator 98, which is used by the programmer to end the program.
4. The PUTRETAIN keyword allows the value that is entered in the ITEM field to be kept in the display. In addition, the ITEM field is defined as an input field by the I in position 38. ITEM is the only input field in these record formats.
5. The ERRMSG keyword identifies the error message that is displayed if indicator 61 is set on in the program that uses this record format.
6. The LOCK keyword prevents the work station user from using the keyboard when the RESPONSE record format is initially displayed.
7. The constants such as "Description", "Price", and "Warehouse Location" describe the fields that are written out by the program.
8. The line and position entries identify where the fields or constants are written on the display.

Figure 9-1 (Part 2 of 2). Example of the Data Description Specifications for a Display Device File

When a record is passed to a program, the fields are arranged in the order in which they are specified in the DDS. The order in which the fields are displayed is based on the display positions (line numbers and position) assigned to the fields in the DDS. Therefore, the order in which the fields are specified in the DDS and the order in which they appear on the screen need not be the same.

Function Key Indicators

The function key indicators KA through KN and KP through KY are valid for a program that contains a WORKSTN file if the associated function key is specified in data description specifications (DDS).

The function key indicators relate to the function keys as follows: Function key indicator KA corresponds to function key 1, KB to function key 2 . . . KX to function key 23, and KY to function key 24.

Function keys are specified in DDS with the CFxx (command function) or CAxx (command attention) keyword. For example, the keyword CF01 allows function key 1 to be used. When function key 1 is pressed, function key indicator KA is set on in the RPG/400 program. If you specify the function key as CF01 (99), both function key indicator KA and indicator 99 are set on in the RPG/400 program. If the work station user presses a function key that is not specified in DDS, OS/400 informs the user that an invalid key was pressed.

If the work station user presses a specified function key, the associated function key indicator in RPG/400 is set on when fields are extracted from the record (move fields logic) and all other function key indicators are set off. If a function key is not pressed, all function key indicators are set off at move fields time. The function key indicators are set off if the user presses the Enter key.

Command Keys

You can specify the command keys Help, Roll Up, Roll Down, Print, Clear, and Home in the data description specifications (DDS) for a display device file with the keywords HELP, ROLLUP, ROLLDOWN, PRINT, CLEAR, and HOME.

Command keys can be processed by an RPG/400 program whenever RPG/400 processes a READ or an EXFMT operation on a record format for which the appropriate keywords are specified in DDS. When the command keys are in effect and a command key is pressed, OS/400 returns control to the RPG/400 program. If a response indicator is specified in the DDS for the command selected, that indicator is set on and all other response indicators that are in effect for the record format and the file are set off.

If a response indicator is not specified in the DDS for a command key, the following happens:

- For the Print key without *PGM, the print function is processed.
- For the Roll Up and Roll Down keys used with subfiles, the displayed subfile rolls up or down, within the subfile. If you try to roll beyond the start or end of a subfile, you get a run-time error.
- For the, Print Key specified with *PGM, Roll Up and Roll Down keys used without subfiles, and for the Clear, Help and Home keys, one of the *STATUS values 1121-1126 is set, respectively, and processing continues.

Processing WORKSTN Files

Figure 9-2 on page 9-6 shows the valid file operation codes for a WORKSTN file.

The EXFMT operation is a combination of a write followed by a read to the same record format. If you define a WORKSTN file on the file description specifications as a full procedural (F in position 16) combined file (C in position 15) that uses externally described data (E in position 19). The EXFMT (execute format) operation code can be used to write and read from the display. The EXFMT operation is a combination of a WRITE followed by a READ to the same record format.

The READ operation is valid for a full procedural combined file or a full procedural input file that uses externally described data or program described data. The READ operation retrieves a record from the display. However, a format must exist at the device before any input operations can occur. This requirement can be satisfied by conditioning an output record with the 1P indicator, by writing the first format to the device from another program, or, if the read is by record format name, by using the keyword INZRCD on the record description in DDS.

The WRITE operation writes a new record to a display and is valid for a combined file or an output file. Output specifications and the EXCPT operation can also be used to write to a WORKSTN file. See the *Languages: RPG/400 Reference* for a complete description of each of these operation codes.

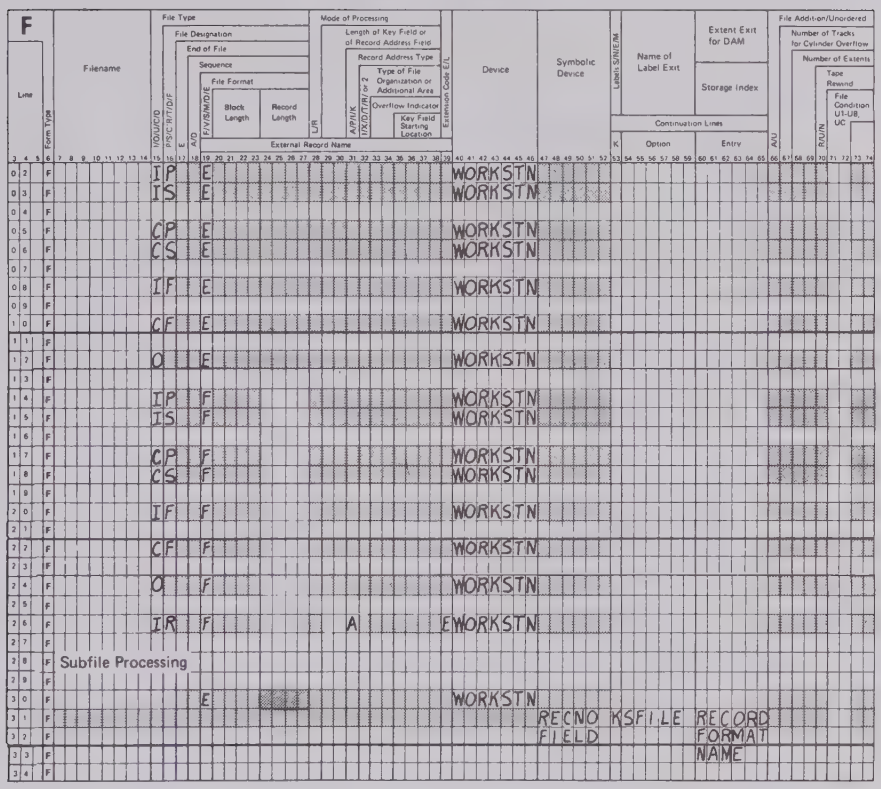
File Description Specifications Positions 15	16	Calculation Specifications Positions 28-32
I	P/S	CLOSE, ACQ, REL, NEXT, POST
C	P/S	WRITE[1], CLOSE, ACQ, REL, NEXT, POST
I	F	READ, OPEN, CLOSE, ACQ, REL, NEXT, POST
C	F	READ, WRITE[1], EXFMT[2], OPEN, CLOSE, ACQ, REL, NEXT, POST
O	Blank	WRITE[1], OPEN, CLOSE, ACQ, REL, POST

[1]The WRITE operation is not valid for a program described file used with a format name.
[2]If the EXFMT operation is used, the file must be externally described (an E in position 19 of the file decription specifications).

Figure 9-2. Valid File Operation Codes for a WORKSTN File

Figure 9-3 on page 9-7 is a processing chart for WORKSTN files.

File Description Specifications



Valid File Operations

- CLOSE
- WRITE, CLOSE
- READ, OPEN, CLOSE
- READ, WRITE, EXFMT, OPEN, CLOSE
- WRITE, OPEN, CLOSE
- CLOSE
- WRITE, CLOSE
- READ, OPEN, CLOSE
- READ, WRITE, OPEN, CLOSE
- WRITE, OPEN, CLOSE
- OPEN, CLOSE
- READC, CHAIN, WRITE, UPDAT (valid only for record defined as a subfile)

Notes:

1. Shaded positions must be blank, and positions without entries are program dependent.
2. WRITE operations to a program described file require a data structure name in the result field; WRITE operations to a program described file that uses a format name on output specifications are not valid.
3. Subfile processing is valid only for an externally described file.

Figure 9-3. Processing Chart for WORKSTN Files

Subfiles

Subfiles can be specified in DDS for a display device file to allow you to handle multiple records of the same type on the display (see Figure 9-4 on page 9-8). A subfile is a group of records that is read from or written to a display device file. For example, a program reads records from a data base file and creates a subfile of output records. When the entire subfile has been written, the program sends the entire subfile to the display device in one write operation. The work station user can change data or enter additional data in the subfile; the program then reads the entire subfile from the display device into the program and processes each record in the subfile individually.

Records that you want to be included in a subfile are specified in the DDS for the file. The number of records that can be included in a subfile must also be specified

in DDS. One file can contain more than one subfile, and up to 12 subfiles can be active concurrently. Two subfiles can be displayed at the same time.

The DDS for a subfile consists of two record formats: a subfile record format and a subfile control record format. The subfile record format contains the field information that is transferred to or from the display file under control of the subfile control record format. The subfile control record format causes the physical read, write, or control operations of a subfile to take place. Figure 9-5 on page 9-10 shows an example of the DDS for a subfile record format, and Figure 9-6 on page 9-11 shows an example of the DDS for a subfile control record format.

For a description of how to use subfile keywords, see the *Programming: Data Description Specifications Reference.*

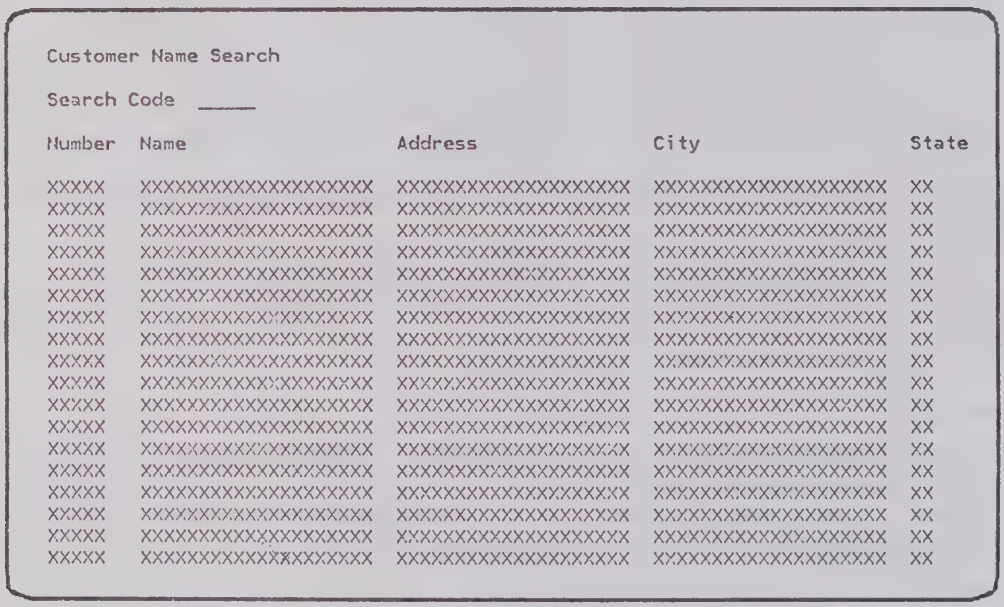


Figure 9-4. Subfile Display

To use a subfile for a display device file in an RPG/400 program, you must specify the SFILE keyword in positions 54 through 59 on a file description specifications continuation line for the WORKSTN file. The SFILE keyword must be specified on a separate continuation line. The WORKSTN file must be an externally described file (E in position 19).

You use positions 60 through 67 of the continuation line to specify the name of the subfile record format (not the control record format). Positions 47 through 52 must specify the name of the field that contains the relative record number to be used in processing the subfile.

In RPG/400, relative record number processing is defined as part of the SFILE definition. The SFILE definition implies a full procedural update file with ADD for the subfile. Therefore, the file operations that are valid for the subfile are not dependent

upon the definition of the main WORKSTN file. That is, the WORKSTN file can be defined as a primary file or a full procedural file.

Use the CHAIN, READC, UPDAT, or WRITE operation codes with the subfile record format to transfer data between the program and the subfile. Use the READ, WRITE, or EXFMT operation codes with the subfile control record format to transfer data between the program and the display device or to process subfile control operations.

Subfile processing follows the rules for relative record number processing. RPG/400 places the relative record number of any record retrieved by a READC operation into the field named in positions 47 through 52 of the file description specifications SFILE continuation line. This field is also used to specify the record number that RPG/400 uses for WRITE operation to the subfile or for output operations that use ADD. The field name specified in positions 47 through 52 must be defined as numeric with zero decimal positions. The field must have enough positions to contain the largest record number for the file (see the SFLSIZ keyword in the *Programming: Data Description Specifications Reference*). The WRITE operation code and the ADD specification on the output specifications require that a relative record number field be specified in positions 47 through 52 of the file description specifications SFILE continuation line.

If a WORKSTN file has an associated subfile, all implicit input operations and explicit calculation operations that refer to the file name are processed against the main WORKSTN file. Any operations that specify a record format name that is not designated as a subfile are processed on the main WORKSTN file.

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-1 UM/050* Printed in U.S.A.

File		Keying Instruction	Graphic										Description	Page of
Programmer	Date		Key											

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (R/F/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (B/A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (B/O/I/I/B/H/M)	Location: Line	Location: Pos	Functions
1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A	**									CUSTOMER NAME SEARCH								REF(DSTREF) 1
	A																		
	A								R		SUBFIL							2	SFL
	A																		TEXT('Subfile Record')
	A										CUST	R					7	3	
	A										NAME	R					7	10	
	A										ADDR	R					7	32	3
	A										CITY	R					7	54	
	A										STATE	R					7	77	
	A																		
	A																		
	A																		

The data description specifications (DDS) for a subfile record format describe the records in the subfile:

1. The attributes for the fields in the record format are contained in the field reference file DSTREF as specified by the REF keyword.
2. The SFL keyword identifies the record format as a subfile.
3. The line and position entries define the location of the fields on the display.

Figure 9-5. Data Description Specifications for a Subfile Record Format

Use of Subfiles

Some typical ways you can make use of subfiles include:

- Display only. The work station user reviews the display.
- Display with selection. The user requests more information about one of the items on the display.
- Modification. The user modifies one or more of the records.
- Input only, with no validity checking. A subfile is used for a data entry function.
- Input only, with validity checking. A subfile is used for a data entry function, but the records are checked.
- Combination of tasks. A subfile can be used as a display with modification, plus the input of new records.

For an example of using a subfile in an RPG/400 program, see "WORKSTN File Examples" on page 9-16.

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754 UM/050* Printed in U.S.A.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Name Type (R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (I/O/I/B/H/M)	Location: Line	Location: Pos	Functions
	A								R		FILCTL								SFLCTL(SUBFIL)
	A		N	70															SFLCLR
	A			70															SFLDSPCTL
	A			71															SFLDSP
	A																		SFLSIZ(15)
	A																		SFLPAG(15)
	A																		TEXT('Subfile Control Record')
	A																		OVERLAY
	A			71															ROLLUP(97 'Continue Search')
	A																		CA01(98 'End of Program')
	A																		HELP(99 'Help Key')
	A																1	2	'Customer Name Search'
	A																3	2	'Search Code'
	A										SRHCOD	R				I	3	14	PUTRETAIN
	A																		
	A																		
	A																5	2	'Number'
	A																5	10	'Name'
	A																5	32	'Address'
	A																5	54	'City'
	A																5	76	'State'
	A																		
	A																		
	A																		
	A																		

Figure 9-6 (Part 1 of 2). Data Description Specifications for a Subfile Control Record Format

The subfile control record format defines the attributes of the subfile, the search input field, constants, and function keys. The keywords you can use indicate the following:

- SFLCTL names the associated subfile (SUBFIL).
- SFLCLR indicates when the subfile should be cleared (when indicator 70 is off).
- SFLDSPCTL indicates when to display the subfile control record (when indicator 70 is on).
- SFLDSP indicates when to display the subfile (when indicator 71 is on).
- SFLSIZ indicates the total number of records to be included in the subfile (15).
- SFLPAG indicates the total number of records in a page (15).
- ROLLUP indicates that indicator 97 is set on in the program when the user presses the Roll Up key.
- HELP allows the user to press the Help key for a displayed message that describes the valid function keys.
- PUTRETAIN allows the value that is entered in the SRHCOD field to be kept in the display.

In addition to the control information, the subfile control record format also defines the constants to be used as column headings for the subfile record format.

Figure 9-6 (Part 2 of 2). Data Description Specifications for a Subfile Control Record Format

Program Described WORKSTN File

You can use a program described WORKSTN file with or without a format name specified on the output specifications. The format name, if specified, refers to the name of a data description specifications record format. This record format describes how the data stream sent from RPG/400 is formatted on the screen. If a format name is used, input and output specifications must be used to describe the input and output records.

You may specify the PASS option on the file description specifications continuation line for a program described WORKSTN file. Positions 60 through 65 must contain *NOIND. The PASS *NOIND indicates that RPG/400 will not additionally pass indicators to data management on output or receive them on input. It is your responsibility to pass indicators by describing them as fields (in the form *INxx, *IN, or *IN,x) in the input or output record. They must be specified in the sequence required by the data description specifications (DDS). You can use the DDS listing to determine this sequence.

Program Described WORKSTN File with a Format Name

The following specifications apply to using a format name for a program described WORKSTN file.

Output Specifications

On the output specifications, you must specify the WORKSTN file name in positions 7 through 14. The format name, which is the name of the DDS record format, is specified as a literal in positions 45 through 54 on the succeeding field description line. K1 through K8 must be specified (right-adjusted) in positions 40 through 43 on the line containing the format name. The K identifies the entry as a length rather than an end position, and the number indicates the length of the format name. For example, if the format name is "CUSPMT", the entry in positions 40 through 43 is K6. (Leading zeros following the K are allowed.) The format name cannot be conditioned (indicators in positions 23 through 31 are not valid).

Output fields must be located in the output record in the same order as defined in DDS; however, the field names do not have to be the same. The end position entries for the fields refer to the end position in the output record passed from RPG/400 to data management, and not to the location of the fields on the screen.

To pass indicators on output, do one of the following:

- Specify the keyword INDARA in the DDS for the WORKSTN file. Do not use the PASS *NOIND option on the file specifications continuation line and do not specify the indicators on the output specifications.
- Specify the PASS *NOIND option on the file specifications continuation line. Specify the indicators in the output specifications as fields in the form *INxx. The indicator fields must precede other fields in the output record, and they must appear in the order specified by the WORKSTN file DDS. You can determine this order from the DDS listing.

Input Specifications

The input specifications describe the record that RPG/400 receives from the display device. The WORKSTN file name must be specified in positions 7 through 14. Input fields must be located in the input record in the same sequence as defined in DDS; however, the field names do not have to be the same. The field location entries refer to the location of the fields in the input record, not to the location of the fields on the screen.

To pass indicators on input, do one of the following:

- Specify the keyword INDARA in the DDS for the WORKSTN file. Do not use the PASS *NOIND option on the file specifications continuation line and do not specify the indicators on the input specifications.
- Specify the PASS *NOIND option on the file specifications continuation line. Specify the indicators in the input specifications as fields in the form *INxx. They must appear in the input record in the order specified by the WORKSTN file DDS. You can determine this order from the DDS listing.

A record identifying indicator should be assigned to each record in the file to identify the record that has been read from the WORKSTN file. A hidden field with a default value can be specified in DDS for the record identification code.

Calculation Specifications

The operation code READ is valid for a program described WORKSTN file that is defined as a combined, full procedural file (see Figure 9-2 on page 9-6). The file name must be specified in factor 2 for this operation. The EXFMT operation is not valid for a program described WORKSTN file. The EXCPT operation can also be used to write to a WORKSTN file.

Additional Considerations

When using a format name with a program described WORKSTN file, you must also consider the following:

- A format must exist at the device before any input operations can take place. This requirement can be satisfied by conditioning an output record with 1P or by writing the first format to the device in another program (for example, in the CL program).
- The name specified in positions 45 through 54 of the output specifications is assumed to be the name of a record format in the DDS that was used to create the file.
- If a Kn specification is present for an output record, it must also be used for any other output records for that file. If a Kn specification is not used for all output records to a file, a run-time error will occur.

For an example of using a format name with a program described WORKSTN file, see "Sample Program 6—Program Described WORKSTN File within FORMAT Name on Output Specifications" on page 9-57.

Program Described WORKSTN File without a Format Name

When a record format name is not used, a program described display device file describes a file containing one record format description with one field. The fields in the record must be described within the program that uses the file.

When you create the display file by using the Create Display File command, the file has the following attributes:

- A variable record length can be specified; therefore, the actual record length must be specified in the using program. (The maximum record length allowed is the screen size minus one.)
- No indicators are passed to or from the program.
- No function key indicators are defined.
- The record is written to the display beginning in position 2 of the first available line.

Input File

For an input file, the input record, which is treated by OS/400 device support as a single input field, is initialized to blanks when the file is opened. The cursor is positioned at the beginning of the field, which is position 2 on the display.

Output File

For an output file, OS/400 device support treats the output record as a string of characters to be sent to the display. Each output record is written as the next sequential record in the file; that is, each record displayed overlays the previous record displayed.

Combined File

For a combined file, the record, which is treated by OS/400 device support as a single field, appears on the screen and is both the output record and the input record. Device support initializes the input record to blanks, and the cursor is placed in position 2.

For more information on program described display device files, see the *Programming: Data Management Guide.*

Multiple Device Files

Any RPG/400 WORKSTN file with at least one of the keywords ID, IND, NUM, or SAVDS specified (on the file specifications continuation line) is a multiple device file. Through a multiple device file, your program may access more than one device.

RPG/400 accesses devices through program devices, which are symbolic mechanisms for directing operations to an actual device. When you create a file (using DDS specifications and commands such as the create file commands), you consider such things as which device is associated with a program device, whether or not a file has a requester device, which record formats will be used to invite devices to respond to a READ-by-file-name operation, and how long this READ operation will wait for a response. For detailed information on the options and requirements for creating a multiple device file, see the chapters on display files and ICF files in the *Communications: Programmer's Guide*, the *Programming: Data Management Guide*, and the manuals you are referred to there.

With multiple device files, you make particular use of the following operation codes:

- In addition to opening a file, the OPEN operation can acquire (at most) one device for a multiple device file. You specify which device when you create the file.
- The ACQ (acquire) operation acquires any other devices for your file.
- The REL (release) operation releases a device from the file.
- The WRITE operation, when used with the DDS keyword INVITE, invites a program device to respond to subsequent read-from-invited- program-devices operations. See the section on inviting a program device in the *Communications: Programmer's Guide* and the *Programming: Data Management Guide.*
- The READ operation either processes a read-from-invited-program-devices operation or a read-from-one-program-device operation. When no NEXT operation is in effect, a program-cycle-read or READ-by-file-name operation waits for input from any of the devices that have been invited to respond (read-from-invited-program-device). Other input and output operations, including a READ-by-file-

name after a NEXT operation, and a READ-by-format-name, process a read-from-one-program-device operation using the program device indicated in a special field. (The field is named in the ID entry of the file specifications continuation lines.)

This device may be the device used on the last input operation, a device you specify, or the requester device. See the sections on reading from invited program devices and on reading from one program device in the *Communications: Programmer's Guide* and the *Programming: Data Management Guide*.

- The NEXT operation specifies which device is to be used in the next READ-by-file-name operation or program-cycle-read.
- The POST operation puts information in the INFDS information data structure. The information may be about a specific device or about the file. (The POST operation is not restricted to use with multiple device files.)

See *Languages: RPG/400 Reference* for details of the RPG/400 operation codes.

On the file specifications continuation line, you may specify several options to control the processing of multiple device files.

- The ID entry specifies the name of a field. The field can contain the name of a program device to which some input and output operations are directed.
- The NUM entry indicates the maximum number of devices that can be acquired for a file.

  By using a value of 1 for NUM, it is possible to get functions associated with a multiple device file for a file that has only one device. For example, Figure 9-40 on page 9-61 illustrates the use of the time-out feature of the READ operation for a multiple device file.
- The SAVDS entry indicates a data structure that is saved and restored for each device acquired to a file. The IND entry indicates a set of indicators to be saved and restored for each device acquired to a file. Before an input operation, the current set of indicators and data structure are saved. After the input operation, RPG/400 restores the indicators and data structure for the device associated with the operation. This may be a different set of indicators or data structure than was available before the input operation.

WORKSTN File Examples

This section illustrates some common work station applications and their RPG/400 coding.

- "Sample Program 1 – Inquiry" on page 9-17 is an example of a basic inquiry program that uses the WORKSTN file in RPG/400.
- "Sample Program 2 – Data Entry with Master Update" on page 9-23 is an example of a data entry with master update program.
- "Sample Program 3 – Maintenance" on page 9-30 is an example of a maintenance program.

- "Sample Program 4—WORKSTN Subfile Processing" on page 9-40 is an example of WORKSTN subfile processing.
- "Sample Program 5—Inquiry by Zip Code and Search on Name" on page 9-48 is an example of an interactive program in which the search of a name field occurs when the work station user enters a zip code and a search string in response to the first display written by the program. This program illustrates one approach to solving the typical problem of trying to identify a customer and determine the correct customer number. In this example, the user knows the zip code and something about the customer name, such as some number of characters.
- "Sample Program 6—Program Described WORKSTN File within FORMAT Name on Output Specifications" on page 9-57 is an example of a program described WORKSTN file with a format name on the output specifications.
- "Sample Program 7—Variable Start Line" on page 9-59 is an example of using the variable start line to determine where a record format will appear on a display.
- "Sample Program 8—Read Operation with Time-Out" on page 9-61 shows how to use READ with a time-out.

Sample Program 1—Inquiry

The following figures illustrate a simple inquiry program using the WORKSTN file:

Table 9-1. List of figures for WORKSTN Inquiry Program

Figure	Contents
Figure 9-7 on page 9-18 and Figure 9-8 on page 9-19	DDS specifications for data base file and display device file
Figure 9-9 on page 9-21	File Description and Calculation specifications
Figure 9-10 on page 9-22	Prompt screen
Figure 9-11 on page 9-23	Customer information screen

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A* CUSTOMER MASTER FILE -- CUSMSTP
A          R CUSREC
A            CUST           5          TEXT('CUSTOMER NUMBER')
A            NAME          20          TEXT('CUSTOMER NAME')
A            ADDR          20          TEXT('CUSTOMER ADDRESS')
A            CITY          20          TEXT('CUSTOMER CITY')
A            STATE          2          TEXT('CUSTOMER STATE')
A            ZIP            5  0       TEXT('CUSTOMER ZIP CODE')
A            SRHCOD         3          TEXT('CUSTOMER NAME SEARCH CODE')
A            CUSTYP         1          TEXT('CUSTOMER TYPE')
A            ARBAL         10  2       TEXT('ACCOUNTS RECEIVABLE BALANCE')

A*  FILE NAME:   CUSMSTL
A*  DESCRIPTION: LOGICAL VIEW OF CUSTOMER MASTER FILE (CUSMSTP)
A*               BY CUSTOMER NUMBER (CUST)
A          R CUSREC                    PFILE(CUSMSTP)
A          K CUST
```

Figure 9-7. DDS for WORKSTN Inquiry Program File CUSMSTP

The data description specifications (DDS) for the data base file used by this program describe one record format: CUSREC. The logical file (CUSMSTL) keyed by customer number is based on the physical file (CUSMSTP), as indicated by the PFILE keyword. Each field in the record format is defined in the physical file (CUSMSTP).

Note: Normally the field attributes, such the as number of decimal positions and the data type, are defined in a field reference file rather than in the DDS for the record format. The attributes are shown on the DDS so you can see what they are.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A*  FILE       : CUSFMT
A*  DESCRIPTION: DISPLAY FILE FOR CUSTOMER MASTER INQUIRY
A                                      DSPSIZ(24 80 *DS3)
A                                      CHGINPDFT(CS)
A                                      CA03(15 'END OF JOB')
A                                      PRINT
A                                      INDARA
A          R CUSHDG
A                                      OVERLAY
A                                    2  4TIME
A                                      DSPATR(HI)
A                                    2 29'Customer Master Inquiry'
A                                      DSPATR(HI)
A                                      DSPATR(UL)
A                                    2 70DATE
A                                      EDTCDE(Y)
A                                      DSPATR(HI)
A          R CUSFTG
A                                   23 20'ENTER - Continue'
A                                      DSPATR(HI)
A                                   23 49'F3 - End Job'
A                                      DSPATR(HI)
A          R CUSPMT
A                                      OVERLAY
A            CUST           5A  I 10 50DSPATR(HI)
A                                      DSPATR(CS)
A  99                                  ERRMSG('Customer Number not Found'
A                                      99)
A                                   10 26'Enter Customer Number:'
A                                      DSPATR(HI)
A          R CUSFLDS
A                                      OVERLAY
A                                    8 25'Name'
A            NAME          20A  O  8 35DSPATR(HI)
A                                   10 25'Address'
A            ADDR          20A  O 10 35DSPATR(HI)
A                                   12 25'City'
A            CITY          20A  O 12 35DSPATR(HI)
A                                   14 25'State'
A            STATE          2A  O 14 35DSPATR(HI)
A                                   14 41'Zip Code'
A            ZIP            5S 00 14 50DSPATR(HI)
A                                   16 25'A/R Balance'
A            ARBAL         10Y 20 16 42DSPATR(HI)
A                                      EDTCDE(J)
A                                    6 25'Customer'
A            CUST           5A  O  6 35DSPATR(HI)
```

Figure 9-8. DDS for WORKSTN Inquiry Program Display Device File CUSFMT

The data description specifications (DDS) for the display device file (CUSFMT) to be used by this program specify file level entries and describe four record formats: CUSHDG, CUSFTG, CUSPMT, and CUSFLDS.

The file level entries define the screen size (DSPSIZ), input defaults (CHGINPDFT), command attention key used to end the program, print key (PRINT), and a separate indicator area (INDARA).

The CUSHDG record format contains the constant "Customer Master Inquiry", which identifies the display. It also contains the keywords TIME and DATE, which will display the current date and time on the screen.

The CUSFTG record format contains the constants "ENTER - Continue" and "F3 - End Job", which describe the processing options.

The CUSPMT record format contains the prompt "Enter Customer Number:" and the input field CUST into which the work-station user enters the customer number. Column separators define the input field on the screen where the user is to enter the customer number. The error message "Customer Number not Found" is also included in this record format. This message is displayed if indicator 99 is set on by the program.

The CUSFLDS record format contains the constants "Name", "Address", "City", "State", "Zip Code", "A/R Balance", and "Customer" that identify the fields to be written out from the program. This record format also describes fields that correspond to these constants. All of these fields are described as output fields because they are filled in by the program; the user does not enter any data into these fields. To enter another customer number, the user presses Enter in response to this record.

In addition to describing the constants, fields and attributes for the screen, the record formats also define the display attributes for the constants and fields and the line numbers and horizontal positions where the constants and fields are to be displayed.

Notice the use of the OVERLAY keyword; the CUSHDG, CUSPMT and CUSFLDS record formats will overlay the CUSFTG record format. The CUSFTG format will remain on the screen when any of the other formats are written to the screen.

Note: Normally the field attributes are defined in a field reference file instead of the DDS for a file. However, they are shown here so you can see the field attributes.

```
*****************************************************************
*                                                               **
*    PROGRAM ID   - CUSTINQ                                     **
*    PROGRAM NAME - CUSTOMER MASTER INQUIRY                     **
*                                                               **
*****************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
FCUSMSTL IF  E           K        DISK
FCUSFMT  CF  E                    WORKSTN
*
C           *IN15     DOWEQ'0'
*
* WRITE HEADING AND FOOTING EXCEPT IF ERROR HAS OCCURRED
* AND PROMPT FOR CUSTOMER NUMBER
C           *IN99     CASEQ'0'       HEADNG
C                     END
C                     EXFMTCUSPMT
*
* IF NOT END OF JOB AND VALID CUSTOMER NUMBER
* DISPLAY CUSTOMER INFORMATION
C           *IN15     IFEQ '0'
C           CUST      CHAINCUSREC              99
C           *IN99     IFEQ '0'
C                     EXFMTCUSFLDS
C                     END
C                     END
C                     END
C                     MOVE '1'       *INLR
*****************************************************************
*    SUBROUTINE - HEADNG                                        **
*    PURPOSE    - DISPLAY HEADING AND FOOTING                   **
*****************************************************************
C           HEADNG    BEGSR
C                     WRITECUSFTG
C                     WRITECUSHDG
C                     ENDSR
```

Figure 9-9. File Description and Calculation Specifications for WORKSTN Inquiry Program

For this program, only the RPG/400 file description and calculation specifications are required. Input and output specifications are not required because both files are externally described files (as indicated by the E in position 19). Both files are described as full procedural files, as indicated by the F in position 16, because the I/O operations are controlled by programmer-specified operation codes. In addition, the K in position 31 of the file description specifications for the CUSMSTL file indicates that the file is processed keys.

The DOWEQ operation will perform a loop until the user presses function key 3 to end the job. Function key 3 sets indicator 15 on, as defined in the DDS. If indicator 15 is on, the loop is ended, the LR indicator is turned on, and the program ends.

The CASEQ operation performs subroutine HEADNG, which writes the heading and footings to the screen. Headings and footings will not be written to the screen when an error has occurred.

The EXFMT operation writes the CUSPMT record to the display. This record prompts the user to enter a customer number. If the user enters a customer number and presses Enter, the same EXFMT operation then reads the record back into the program.

If the user does not end the job, the CHAIN operation retrieves a record from the CUSMSTL file. Note that the record format name (CUSREC) is specified for the CHAIN operation rather than the file name. If the record is not found, indicator 99 is set on and the program loops back to display the CUSPMT record again. The message "Customer Number not Found" is displayed, the ERRMSG keyword in the DDS is conditioned by indicator 99, and the keyboard is locked. The user must press the Reset key in response to this message to unlock the keyboard. The user can then enter another customer number.

If the CHAIN operation retrieves a record from the CUSMSTL file, the EXFMT operation writes the record CUSFLDS to the display work station. This record contains the customer's name, address information, and accounts receivable balance.

The user then presses Enter, and the program loops back to the EXFMT operation and writes record CUSPMT to the display work station. The user can enter another customer number or end the program.

This is the initial display written to the display WORKSTN by the EXFMT:

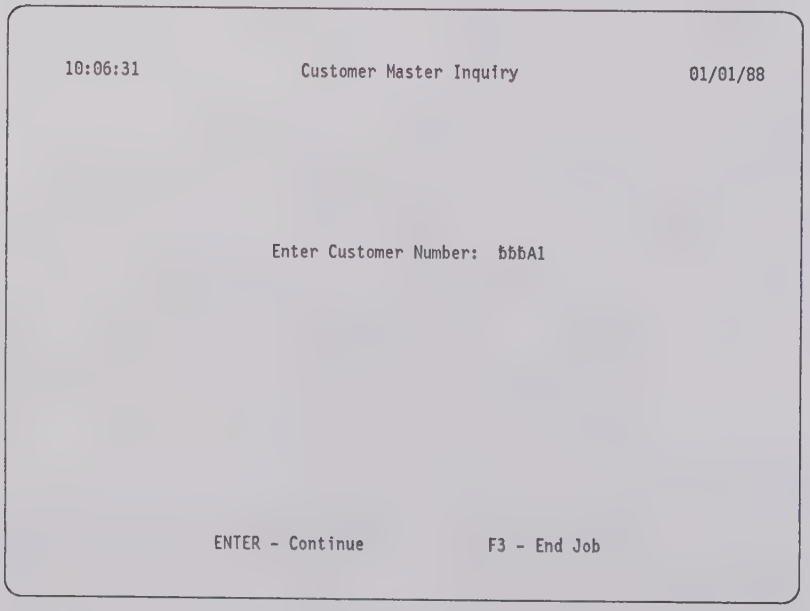


Figure 9-10. Customer Inquiry Prompt Screen

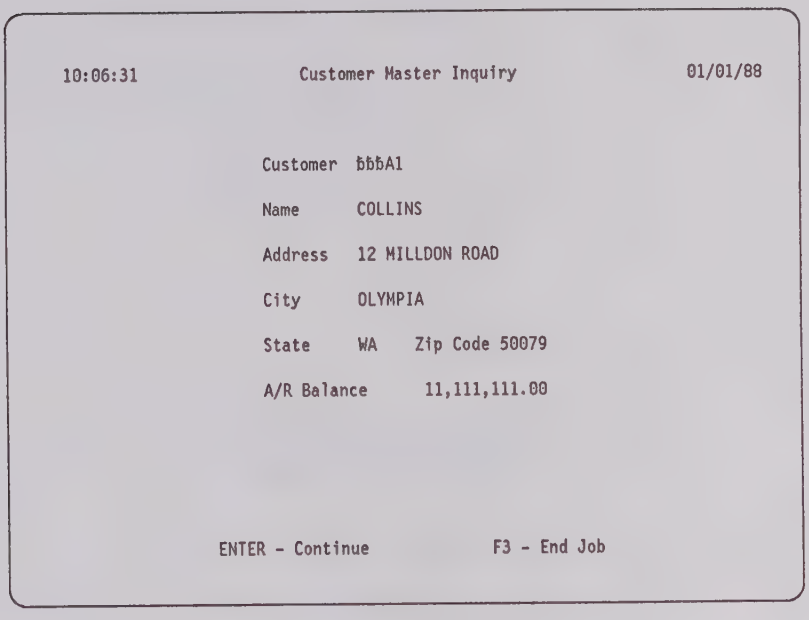


Figure 9-11. Customer Inquiry Information Screen

This display appears if a record is found in the CUSTMSTL file with the same customer number that was entered by the user in response to the first display.

Sample Program 2—Data Entry with Master Update

The following figures illustrate a Data Entry program that prompts the user, updates a master record, and writes a transaction file:

Table 9-2. List of figures for WORKSTN Data Entry Program

Figure	Contents
Figure 9-12 on page 9-24 and Figure 9-13 on page 9-25	DDS specifications for master file, transaction file and display device file
Figure 9-14 on page 9-27	File Description and Calculation specifications
Figure 9-15 on page 9-28	Prompt screen
Figure 9-16 on page 9-29	Display of Current Information
Figure 9-17 on page 9-29	Updated Screen

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A* PARTS MASTER FILE -- PARTMST
A          R MSTREC
A            PART#          5S 0       TEXT('PART NUMBER')
A            DESCRP        20          TEXT('DESCRIPTION')
A            ISSUE          7S 0       TEXT('QTY ISSUED')
A            RECPT          7S 0       TEXT('QTY RECEIVED')
A            ONHAND         9S 0       TEXT('QTY ON HAND')
A            DTLUPD         6S 0       TEXT('DATE LAST UPDATE')
A          K PART#


A* PARTS TRANSACTION FILE -- TRNFIL
A          R TRNREC
A            PARTNO         5S 0       TEXT('PART NUMBER')
A            QTYISS         7S 0       TEXT('QTY ISSUED')
A            QTYREC         7S 0       TEXT('QTY RECEIVED')
A            DATE           6S 0       TEXT('CURRENT DATE')
```

Figure 9-12. DDS for Data Entry/Update Master File and Transaction File

The data description specifications (DDS) for the data base files used by this program describe two record formats: MSTREC and TRNREC. The master file (PARTMST) is a keyed physical file; the transaction file (TRNFIL) is a sequential file.

Note: Normally the field attributes, such as the number of decimal positions and the data type, are defined in a field reference file rather than in the DDS for the record format. The attributes are shown on the DDS so you can see what the field attributes are.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A*  FILE       : PRTUPD
A*  DESCRIPTION: TRANSACTION AND MASTER FILE UPDATE
A*****************************************************************
A                                     CHGINPDFT(CS)
A                                     PRINT(QSYSPRT)
A                                     INDARA
A                                     CA03(03 'END OF JOB')
A          R PROMPT
A                                   2  4TIME  DSPATR(HI)
A                                   2 28'PART TRANSACTION ENTRY'
A                                      DSPATR(HI UL)
A                                   2 70DATE  EDTCDE(Y) DSPATR(HI)
A                                   6  4'Enter Part Number'
A                                      DSPATR(HI)
A            PART#     R     Y  I  6 23REFFLD(PART# PARTMST)
A                                      DSPATR(CS) CHECK(RB)
A  61                                  ERRMSG('PART # NOT FOUND' +
A                                      61)
A                                  23  6'ENTER - Continue'
A                                      DSPATR(HI)
A                                  23 29'F3 - End Job'
A                                      DSPATR(HI)
A          R TRNFMT
A                                      CA12(12 'CANCEL TRANS')
A                                   2  4TIME  DSPATR(HI)
A                                   2 28'PART TRANSACTION ENTRY'
A                                      DSPATR(HI UL)
A                                   2 70DATE  EDTCDE(Y)  DSPATR(HI)
A                                   6 10'Part Number'
A                                      DSPATR(HI)
A            PART#     R     Y  0  6 23REFFLD(PART# PARTMST)
A            DESCRP    R        0  7 23REFFLD(DESCRP PARTMST)
A                                   9 10'Qty On Hand'
A            ONHAND    R     Y  0  9 23REFFLD(ONHAND PARTMST)
A                                      DSPATR(HI) EDTCDE(Z)
A                                  11 10'Qty Issued '
A            QTYISS    R     Y  B 11 25REFFLD(QTYISS TRNFIL)
A                                      CHECK(RB)
A                                      DSPATR(HI CS)
A                                  13 10'Qty Received'
A            QTYREC    R     Y  B 13 25REFFLD(QTYREC TRNFIL)
A                                      CHECK(RB) DSPATR(HI CS)
A                                  23  6'ENTER - Continue'
A                                      DSPATR(HI)
A                                  23 29'F3 - End Job'
A                                      DSPATR(HI)
A                                  23 46'F12 - Cancel Transaction'
A                                      DSPATR(HI)
```

Figure 9-13. DDS for Data Entry/Update PRTUPD Display Device File

The DDS for the PRTUPD display device file contains two record formats: PROMPT and TRNFMT. The PROMPT record prompts for the part number to be processed. If the part is not found, an error message is displayed. The TRNFMT record is used to enter

issue and receipt quantities. The fields are defined as output/input (B in position 38) and output (O in position 38).

Function key 3 has been defined at the file level and is valid for all record formats. Function key 12 is defined at the record level for the TRNFMT record format and is not valid for any other format.

```
*********************************************************************
*    PROGRAM ID   - DTAENT                                          **
*    PROGRAM NAME - TRANSACTION MAINTENANCE                         **
*    THIS PROGRAM PERFORMS THE FOLLOWING FUNCTIONS;                 **
*    - ADDS NEW TRANSACTION RECORDS TO THE FILE TRNFIL              **
*    - UPDATES PART MASTER FILE PARTMST                             **
*********************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
FPARTMST UF  E           K       DISK
FTRNFIL  O   E           K       DISK
FPRTUPD  CF  E                   WORKSTN
*************************************************
*      MAINLINE
*************************************************
C                   EXFMTPROMPT
*
C          *IN03    DOWEQ'0'
*
C          PART#    CHAINMSTREC             61
C          *IN61    CASEQ'0'       NXTSCN
C                   END
*
C          *IN03    IFEQ '0'
C                   EXFMTPROMPT
C                   END
*
C                   END
C                   MOVE '1'       *INLR
*********************************************************************
*    SUBROUTINE - NXTSCN                                            **
*    PURPOSE    - ADD PART TRANSACTIONS                             **
*********************************************************************
C          NXTSCN   BEGSR
C                   EXFMTTRNFMT
C          *IN03    IFEQ '0'
C          *IN12    ANDEQ'0'
C                   ADD  QTYISS    ISSUE
C                   ADD  QTYREC    RECPT
C                   ADD  QTYREC    ONHAND
C                   SUB  QTYISS    ONHAND
C                   Z-ADDUDATE     DTLUPD
C                   Z-ADDUDATE     DATE
C                   UPDATMSTREC
C                   WRITETRNREC
C                   Z-ADD*ZERO     QTYISS
C                   Z-ADD*ZERO     QTYREC
C                   ELSE
C                   EXCPTRLS
C                   END
C                   ENDSR
OMSTREC  E              RLS
```

Figure 9-14. File Description Specification and Calculation Specification for Data Entry/Update Program

This program (data entry with master update) prompts the user for a transaction, updates a master record, and writes a transaction record.

The program has two disk files (PARTMST and TRNFIL) and one WORKSTN file (PRTUPD). The program begins by prompting the work-station user for a part number. The user can press function key 3, which is associated with indicator 03 in the DDS, to end the program.

The CHAIN operation retrieves the master record. If the record is not found, an error message is displayed; otherwise, the record format TRNFMT is displayed. The user can press function key 12 to cancel the transaction; the master record is released, and the PROMPT record format is redisplayed. The user can press function key 3 to end the program, or the user can process the transaction. When the user presses ENTER after entering issue and/or receipt quantities, the master file (PARTMST) is updated with the current date, new on hand quantity, issues and receipts, and the transaction is added to the transaction file (TRNFIL)

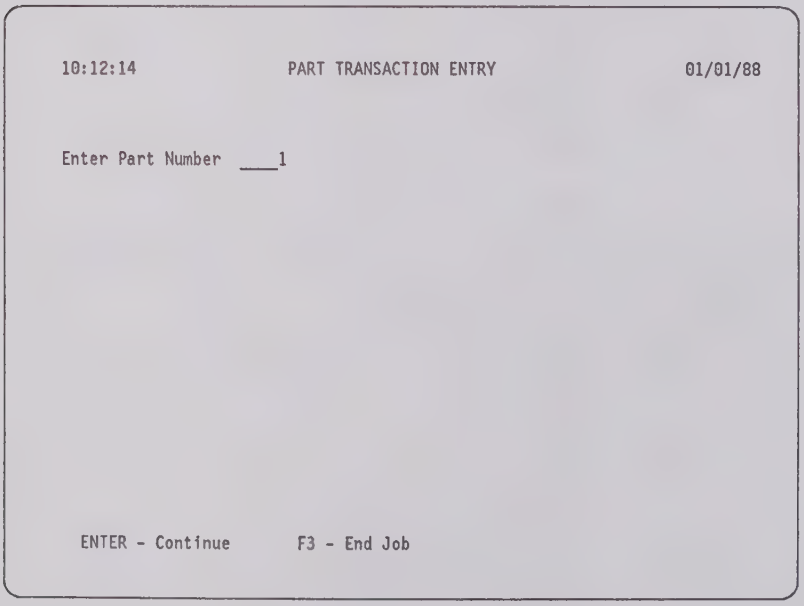


Figure 9-15. Prompt Screen for Data Entry/Update Program

The work-station user responds to the prompts on the first screen by entering a part number.

```
10:12:20              PART TRANSACTION ENTRY                01/01/88


      Part Number  00001
                   ALPHABC

      Qty On Hand       50

      Qty Issued        20

      Qty Received      50




   ENTER - Continue       F3 - End Job     F12 - Cancel Transaction
```

Figure 9-16. TRNFMT Screen

Because part number 1 is in the Customer Master File, the program displays the record for that part.

```
10:12:38              PART TRANSACTION ENTRY                01/01/88


      Part Number  00001
                   ALPHABC

      Qty On Hand        80

      Qty Issued    0000000

      Qty Received  0000000




   ENTER - Continue       F3 - End Job     F12 - Cancel Transaction
```

Figure 9-17. TRNFIL Screen

The work-station user can press Enter to continue or Function Key 12 to cancel the transaction.

Sample Program 3—Maintenance

The following figures illustrate a simple inquiry program using the WORKSTN file:

Table 9-3. List of figures for WORKSTN Inquiry Program

Figure	Contents
Figure 9-18 and Figure 9-19 on page 9-31	DDS specifications for master file and display device file
Figure 9-20 on page 9-34	File Description and Calculation specifications
Figure 9-21 on page 9-37	Display mode prompt screen
Figure 9-22 on page 9-38	Add mode prompt screen
Figure 9-23 on page 9-39	Update mode prompt screen
Figure 9-24 on page 9-40	Delete mode prompt screen

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A* CUSTOMER MASTER FILE -- CUSTMSTR
A          R CSTMST
A            CUST#          5S 0       TEXT('CUSTOMER NUMBER')
A            CSTNAM        20          TEXT('CUSTOMER NAME')
A            CSTAD1        20          TEXT('CUSTOMER ADDRESS')
A            CSTAD2        20          TEXT('CUSTOMER ADDRESS')
A            CSTCTY        20          TEXT('CUSTOMER CITY')
A            CSTSTE         2          TEXT('CUSTOMER STATE')
A            CSTZIP         5S 0       TEXT('CUSTOMER ZIP CODE')
A          K CUST#
```

Figure 9-18. DDS for Maintenance Program Master File

The data description specifications (DDS) for the data base file used by this program describe one record format: CSTMST. Each field in the record format is described, and the CUST# field is identified as the key field for the record format.

Note: Normally the field attributes, such as number of decimal positions and data type, are defined in a field reference file rather than in the DDS for the record format. The attributes are shown on the DDS so you can see what the field attributes are.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A*  FILE       : CSTENT
A*  DESCRIPTION: DISPLAY FILE FOR CUSTOMER MASTER INQUIRY  EX17-10
A*
A* SELECT OPTION SCREEN
A*********************************************************************
A                                      REF(CUSTMSTR)
A                                      CHGINPDFT(CS)
A                                      PRINT(QSYSPRT)
A                                      INDARA
A          R HDRSCN
A                                      TEXT('PROMPT FOR CUST NUMBER')
A                                      CA03(03 'END OF INQUIRY')
A                                      CA05(05 'ADD MODE')
A                                      CA06(06 'UPDATE MODE')
A                                      CA07(07 'DELETE MODE')
A                                      CA08(08 'DISPLAY MODE')
A            MODE           8A  O  1  4DSPATR(HI)
A                                     1 13'MODE'
A                                       DSPATR(HI)
A                                     2  4TIME
A                                       DSPATR(HI)
A                                     2 28'CUSTOMER FILE MAINTENANCE'
A                                       DSPATR(HI RI)
A                                     2 70DATE
A                                       EDTCDE(Y)
A                                       DSPATR(HI)
A            CUST#      R     Y  I 10 25DSPATR(CS)
A                                       CHECK(RZ)
A  51                                   ERRMSG('CUSTOMER ALREADY ON +
A                                       FILE' 51)
A  52                                   ERRMSG('CUSTOMER NOT ON FILE' +
A                                       52)
A                                    10 33'<--Enter Customer Number'
A                                       DSPATR(HI)
A                                    23  4'F3 End Job'
A                                    23 21'F5 Add'
A                                    23 34'F6 Update'
A                                    23 50'F7 Delete'
A                                    23 66'F8 Display'
A          R CSTINQ                      TEXT('DISPLAY CUST INFO')
A                                        CA12(12 'PREVIOUS SCREEN')
A            MODE           8   0  1  4DSPATR(HI)
A                                     1 13'MODE'
A                                       DSPATR(HI)
A                                     2  4TIME
A                                       DSPATR(HI)
A                                     2 28'CUSTOMER FILE MAINTENANCE'
A                                       DSPATR(HI RI)
A                                     2 70DATE
A                                       EDTCDE(Y)
A                                       DSPATR(HI)
```

Figure 9-19 (Part 1 of 2). DDS for Display Device File for Customer Master Inquiry

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A                                      4 14'Customer:'
A                                        DSPATR(HI UL)
A            CUST#     R         0  4 25DSPATR(HI)
A            CSTNAM    R         B  6 25DSPATR(CS)
A  04                                    DSPATR(PR)
A            CSTAD1    R         B  7 25DSPATR(CS)
A  04                                    DSPATR(PR)
A            CSTAD2    R         B  8 25DSPATR(CS)
A  04                                    DSPATR(PR)
A            CSTCTY    R         B  9 25DSPATR(CS)
A  04                                    DSPATR(PR)
A            CSTSTE    R         B 10 25DSPATR(CS)
A  04                                    DSPATR(PR)
A            CSTZIP    R         B 10 40DSPATR(CS)
A                                        EDTCDE(Z)
A  04                                    DSPATR(PR)
A                                     23  2'F12 Cancel'
A            MODE1          8    0 23 20
A          R CSTBLD                      TEXT('ADD CUST RECORD')
A                                        CA12(12 'PREVIOUS SCREEN')
A            MODE           8    0  1  4DSPATR(HI)
A                                      1 13'MODE'     DSPATR(HI)
A                                      2  4TIME
A                                        DSPATR(HI)
A                                      2 28'CUSTOMER FILE MAINTENANCE'
A                                        DSPATR(HI RI)
A                                      2 70DATE
A                                        EDTCDE(Y)
A                                        DSPATR(HI)
A                                      4 14'Customer:' DSPATR(HI UL)
A            CUST#     R         0  4 25DSPATR(HI)
A                                      6 20'Name'     DSPATR(HI)
A            CSTNAM    R         I  6 25
A                                      7 17'Address'  DSPATR(HI)
A            CSTAD1    R         I  7 25
A                                      8 17'Address'  DSPATR(HI)
A            CSTAD2    R         I  8 25
A                                      9 20'City'     DSPATR(HI)
A            CSTCTY    R         I  9 25
A                                     10 19'State'    DSPATR(HI)
A            CSTSTE    R         I 10 25
A                                     10 36'Zip'      DSPATR(HI)
A            CSTZIP    R     Y   I 10 40
A                                     23  2'F12 Cancel Addition'
```

Figure 9-19 (Part 2 of 2). DDS for Display Device File for Customer Master Inquiry

The DDS for the CSTENT display device file contains three record formats: HDRSCN, CSTINQ, and CSTBLD. The HDRSCN record prompts for the customer number and the mode of processing. The CSTINQ record is used for the Update, Delete, and Display modes. The fields are defined as output/input (B in position 38). The fields are protected when Display or Delete mode is selected (DSPATR(PR)). The CSTBLD record provides only input fields (I in position 38) for a new record.

The CUSHDG record format contains the constant "Customer Master Inquiry"; the ERRMSG keyword defines the messages to be displayed if an error occurs. The CA keywords define the function keys that can be used and associate the function keys with indicators in the RPG program.

```
 ********************************************************************
 *    PROGRAM ID   - CUSTMNT                                        **
 *    PROGRAM NAME - CUSTOMER MASTER MAINTENANCE                    **
 *    THIS PROGRAM ADDS, UPDATES, DELETES AND DISPLAYS              **
 *    CUSTOMER RECORDS IN THE CUSTOMER MASTER FILE.                 **
 ********************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
FCUSTMSTRUF  E           K       DISK                      A
FCSTENT  CF  E                   WORKSTN
 *
C           CSTKEY    KLIST
C                     KFLD           CUST#
 ****************************************************
 *       MAINLINE
 ****************************************************
C                     MOVE 'DISPLAY 'MODE
C                     EXFMTHDRSCN
 *
C           *IN03     DOWEQ'0'
C                     EXSR SETMOD
 *
C           CUST#     IFNE *ZERO
C           MODE      CASEQ'ADD'     ADDSUB
C           MODE      CASEQ'UPDATE'  UPDSUB
C           MODE      CASEQ'DELETE'  DELSUB
C           MODE      CASEQ'DISPLAY' INQSUB
C                     END
C                     END
 *
C                     EXFMTHDRSCN
C                     END
C                     MOVE '1'       *INLR
 ********************************************************************
 *    SUBROUTINE - ADDSUB                                           **
 *    PURPOSE    - ADD NEW CUSTOMER TO FILE                         **
 ********************************************************************
C           ADDSUB    BEGSR
C           CSTKEY    CHAINCSTMST              50
C           *IN50     IFEQ '0'
C                     MOVE '1'       *IN51
C                     ELSE
C                     MOVE '0'       *IN51
C                     MOVE *BLANK    CSTNAM
C                     MOVE *BLANK    CSTAD1
C                     MOVE *BLANK    CSTAD2
C                     MOVE *BLANK    CSTCTY
C                     MOVE *BLANK    CSTSTE
C                     Z-ADD*ZERO     CSTZIP
C                     EXFMTCSTBLD
C           *IN12     IFEQ '0'
C                     WRITECSTMST
C                     END
C                     END
C                     ENDSR
```

Figure 9-20 (Part 1 of 3). File Description and Calculation Specifications for Maintenance Program

```
*****************************************************************
*    SUBROUTINE - UPDSUB                                        **
*    PURPOSE    - UPDATE CUSTOMER MASTER RECORD                 **
*****************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
C           UPDSUB    BEGSR
C                     MOVE '0'       *IN04
C           CSTKEY    CHAINCSTMST              52
C           *IN52     IFEQ '0'
C                     EXFMTCSTINQ
C           *IN12     IFEQ '0'
C                     UPDATCSTMST
C                     ELSE
C                     EXCPTRLS
C                     END
C                     END
C                     ENDSR
*****************************************************************
*    SUBROUTINE - DELSUB                                        **
*    PURPOSE    - DELETE CUSTOMER MASTER RECORD                 **
*****************************************************************
C           DELSUB    BEGSR
C                     MOVE '1'       *IN04
C           CSTKEY    CHAINCSTMST              52
C           *IN52     IFEQ '0'
C                     EXFMTCSTINQ
C           *IN12     IFEQ '0'
C                     DELETCSTMST
C                     ELSE
C                     EXCPTRLS
C                     END
C                     END
C                     ENDSR
*****************************************************************
*    SUBROUTINE - INQSUB                                        **
*    PURPOSE    - DISPLAY CUSTOMER MASTER RECORD                **
*****************************************************************
C           INQSUB    BEGSR
C                     MOVE '1'       *IN04
C           CSTKEY    CHAINCSTMST              52
C           *IN52     IFEQ '0'
C                     EXFMTCSTINQ
C                     EXCPTRLS
C                     END
C                     ENDSR
```

Figure 9-20 (Part 2 of 3). File Description and Calculation Specifications for Maintenance Program

```
*****************************************************************
*    SUBROUTINE - SETMOD                                        **
*    PURPOSE    - SET MAINTENANCE MODE                          **
*****************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
C           SETMOD    BEGSR
C           *IN05     IFEQ '1'
C                     MOVE 'ADD     'MODE
C                     MOVE MODE      MODE1
C                     ELSE
C           *IN06     IFEQ '1'
C                     MOVE 'UPDATE  'MODE
C                     MOVE MODE      MODE1
C                     ELSE
C           *IN07     IFEQ '1'
C                     MOVE 'DELETE  'MODE
C                     MOVE MODE      MODE1
C                     ELSE
C           *IN08     IFEQ '1'
C                     MOVE 'DISPLAY 'MODE
C                     MOVE MODE      MODE1
C                     ELSE
C                     END
C                     END
C                     END
C                     END
C                     ENDSR
OCSTMST  E                RLS
```

Figure 9-20 (Part 3 of 3). File Description and Calculation Specifications for Maintenance Program

This program maintains a customer master file for additions, changes, and deletions. The program can also be used for inquiry.

The program first sets the default (display) mode of processing and displays the customer maintenance prompt screen. The work-station user can press function key 3, which turns on indicator 03, to request end of job. Otherwise, the user enters a customer number and presses Enter. The user can change the mode of processing by pressing function key 5 (ADD), 6 (UPDATE), 7 (DELETE), or 8 (DISPLAY).

To add a new record to the file, the program uses the customer number as the search argument to chain to the master file. If the record does not exist in the file, the program displays the CSTBLD screen to allow the user to enter a new customer record. If the record is already in the file, an error message is displayed. The user can press function key 12, which sets on indicator 12, to cancel the add operation and release the record. Otherwise, the user enters information for the new customer record in the input fields and writes the new record to the master file.

To update, delete, or display an existing record, the program uses the customer number as the search argument to chain to the master file. If a record for that customer exists in the file, the program displays the customer file inquiry screen (CSTINQ). If the record is not in the file, an error message is displayed. If the mode of processing is display or delete, the input fields are protected from modification. Otherwise, the user can enter new information in the input fields to update the cus-

tomer record. The user can press function key 12, which sets on indicator 12, to cancel the update or delete operation, and release the record. Display mode automatically releases the record when the enter key is pressed.

```
DISPLAY  MODE
10:09:01              CUSTOMER FILE MAINTENANCE              01/01/88




                    00001   <--Enter Customer Number




F3 End Job       F5 Add       F6 Update       F7 Delete       F8 Display
```

The work-station user responds to the prompt by entering customer number 00001 to display the customer record.

```
DISPLAY  MODE
10:09:11                CUSTOMER FILE MAINTENANCE              01/01/88

          Customer:  00001

                     SMITH, JOE
                     SUITE 20000
                     QUEEN STREET
                     PORTLAND
                     OR             99999




F12 Cancel DISPLAY
```

Figure 9-21. Display Mode Screens for Maintenance Program

Because the customer record for customer number 00001 exists in the Master File, the data is displayed.

```
ADD      MODE
10:09:20              CUSTOMER FILE MAINTENANCE              01/01/88




                      00009   <--Enter Customer Number




F3 End Job       F5 Add       F6 Update       F7 Delete       F8 Display
```

The work-station user responds to the add prompt by entering a new customer number.

Figure 9-22 (Part 1 of 2). Add Mode Screens for Maintenance Program

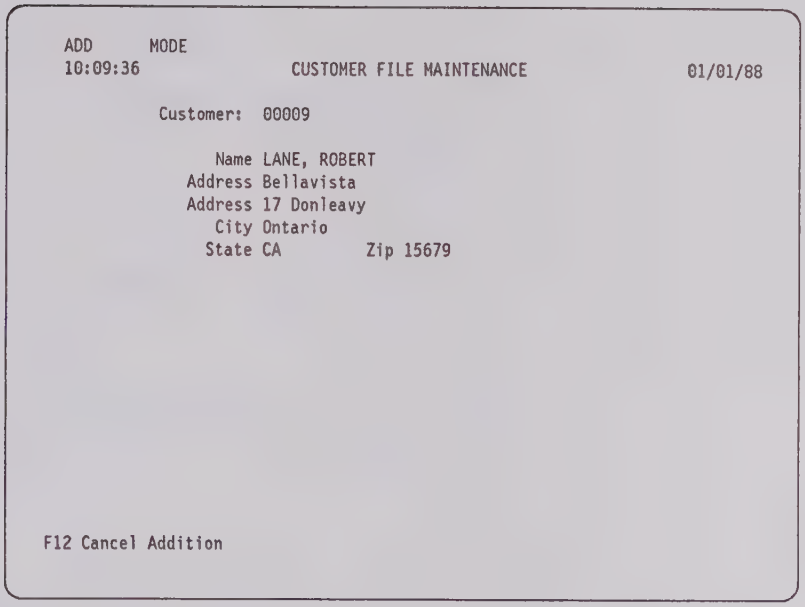


Figure 9-22 (Part 2 of 2). Add Mode Screens for Maintenance Program

A new customer is added to the Customer Master File.

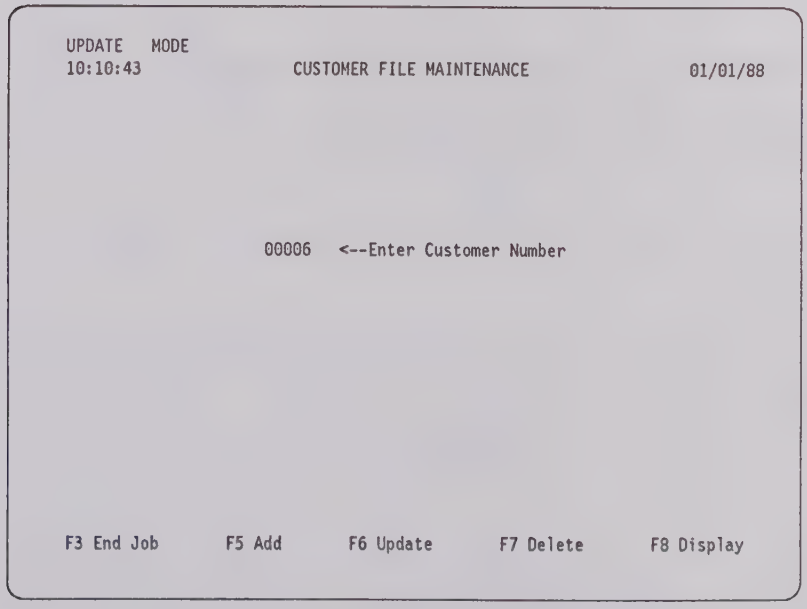


Figure 9-23. Update Mode Screens for Maintenance Program

The work-station user responds to the update prompt by entering a customer number.

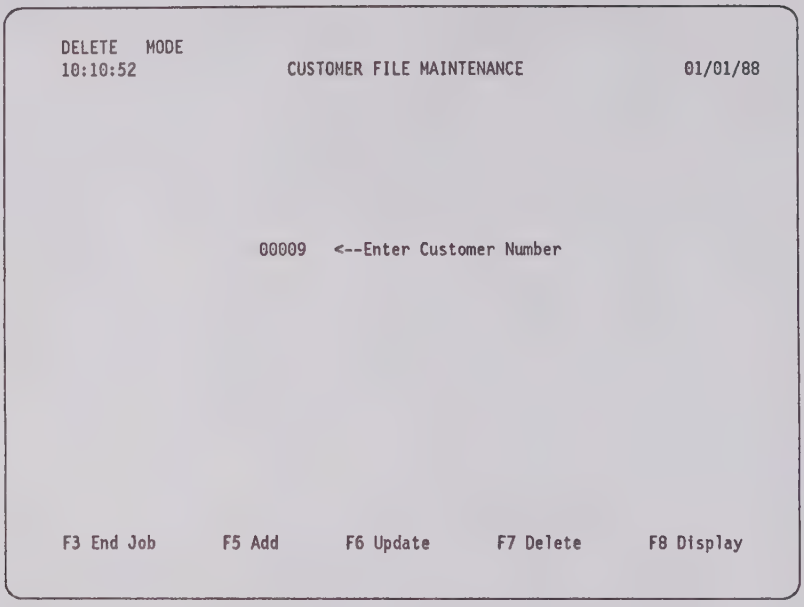


Figure 9-24. Delete Mode Screens for Maintenance Program

The work-station user responds to the delete prompt by entering a new customer number.

Sample Program 4—WORKSTN Subfile Processing

The following figures illustrate a WORKSTN file:

Table 9-4. List of figures for WORKSTN Subfile Processing	
Figure	**Contents**
Figure 9-25 on page 9-41 and Figure 9-26 on page 9-42	DDS specifications for master file and display device file
Figure 9-27 on page 9-44	File Description and Calculation specifications
Figure 9-28 on page 9-47	Prompt screen
Figure 9-29 on page 9-47	Display screen

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A* CUSTOMER MASTER FILE -- CUSMSTP
A          R CUSREC
A            CUST           5          TEXT('CUSTOMER NUMBER')
A            NAME          20          TEXT('CUSTOMER NAME')
A            ADDR          20          TEXT('CUSTOMER ADDRESS')
A            CITY          20          TEXT('CUSTOMER CITY')
A            STATE          2          TEXT('CUSTOMER STATE')
A            ZIP            5  0       TEXT('CUSTOMER ZIP CODE')
A            SRHCOD         3          TEXT('CUSTOMER NAME SEARCH CODE')
A            CUSTYP         1          TEXT('CUSTOMER TYPE')
A            ARBAL         10  2       TEXT('ACCOUNTS RECEIVABLE BALANCE')

A*  FILE NAME:   CUSZIPL
A*  DESCRIPTION: LOGICAL VIEW OF CUSTOMER MASTER FILE (CUSMSTP)
A*               BY CUSTOMER ZIP CODE (ZIP)
A          R CUSREC                    PFILE(CUSMSTP)
A            ZIP       R
A            NAME      R
A            ARBAL     R
A          K ZIP
```

Figure 9-25. DDS for WORKSTN Subfile Processing Program Master File

The data description specifications (DDS) for the data base file used by this program describe one record format: CUSREC. The logical file (CUSZIPL) keyed by zip code is based on the physical file (CUSMSTP), as indicated by the PFILE keyword. The record format generated by the logical file will include only those fields specified in the logical file DDS. All other fields will be excluded.

Note: Normally the field attributes, such as number of decimal positions and data type, are defined in a field reference file rather than in the DDS for the record format. The attributes are shown on the DDS so you can see what the field attributes are.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A*  FILE       : CUSSRC
A*  DESCRIPTION: DISPLAY CUSTOMER MASTER BY ZIP CODE
A                                     REF(CUSMSTP)
A                                     CHGINPDFT(CS)
A                                     PRINT(QSYSPRT)
A                                     INDARA
A                                     CA03(03 'END OF JOB')
A          R HEAD
A                                     OVERLAY
A                                   2  4TIME
A                                     DSPATR(HI)
A                                   2 28'CUSTOMER FILE SEARCH'
A                                     DSPATR(HI RI)
A                                   2 70DATE
A                                     EDTCDE(Y)
A                                     DSPATR(HI)
A          R FOOT1
A                                  23  6'ENTER - Continue'
A                                     DSPATR(HI)
A                                  23 29'F3 - End Job'
A                                     DSPATR(HI)
A          R FOOT2
A                                  23  6'ENTER - Continue'
A                                     DSPATR(HI)
A                                  23 29'F3 - End Job'
A                                     DSPATR(HI)
A                                  23 47'F4 - RESTART ZIP CODE'
A                                     DSPATR(HI)
```

Figure 9-26 (Part 1 of 2). DDS for WORKSTN Subfile Processing Program Display Device File

```
A          R PROMPT
A                                      OVERLAY
A                                    4  4'Enter Zip Code'
A                                       DSPATR(HI)
A            ZIP       R     Y  I  4 19DSPATR(CS)
A                                       CHECK(RZ)
A  61                                   ERRMSG('ZIP CODE NOT FOUND' +
A                                       61)
A          R SUBFILE                    SFL
A            NAME      R             9  4
A            ARBAL     R             9 27EDTCDE(J)
A          R SUBCTL                     SFLCTL(SUBFILE)
A  55                                   SFLCLR
A N55                                   SFLDSPCTL
A N55                                   SFLDSP
A                                       SFLSIZ(13)
A                                       SFLPAG(13)
A                                       ROLLUP(95 'ROLL UP')
A                                       OVERLAY
A                                       CA04(04 'RESTART ZIP CDE')
A                                    4  4'Zip Code'
A            ZIP       R        O  4 14DSPATR(HI)
A                                    7  4'Customer Name'
A                                       DSPATR(HI UL)
A                                    7 27'A/R Balance'
A                                       DSPATR(HI UL)
```

Figure 9-26 (Part 2 of 2). DDS for WORKSTN Subfile Processing Program Display Device File

The DDS for the CUSSRC display device file contains six record formats: HEAD, FOOT1, FOOT2, PROMPT, SUBFILE, and SUBCTL.

The PROMPT record format requests the user to enter a zip code. If the zip code is not found in the file, an error message is displayed. The user can press function key 3, which sets on indicator 03, to end the program.

The SUBFILE record format must be defined immediately preceding the subfile control record format (SUBCTL). The subfile record format, which is defined with the keyword SFL, describes each field in the record, and specifies the location where the first record is to appear on the display (here, on line 9).

The subfile control record format contains the following unique keywords:

- SFLCTL identifies this format as the control record format and names the associated subfile record format.
- SFLCLR describes when the subfile is to be cleared of existing records (when indicator 55 is on). This keyword is needed for additional displays.
- SFLDSPCTL indicates when to display the subfile control record format (when indicator 55 is off).
- SFLDSP indicates when to display the subfile (when indicator 55 is off).
- SFLSIZ specifies the total size of the subfile. In this example, the subfile size is 13 records that are displayed on lines 9 through 21.

- SFLPAG defines the number of records on a page. In this example, the page size is the same as the subfile size.
- ROLLUP indicates that indicator 95 is set on in the program when the rollup function is used.

The OVERLAY keyword defines this subfile control record format as an overlay format. This record format can be written without OS/400 erasing the screen first. Function key 3 is valid for repeating the search with the same zip code. (This use of function key 3 allows a form of rolldown.)

```
 ********************************************************************
 *                                                                  **
 *    PROGRAM ID   - CUSTSFL                                        **
 *    PROGRAM NAME - CUSTOMER MASTER SEARCH                         **
 *      THIS PROGRAM DISPLAYS THE CUSTOMER MASTER FILE              **
 *      BY ZIP CODE                                                 **
 ********************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
FCUSZIPL IF  E           K        DISK
FCUSSRC  CF  E                    WORKSTN
F                                        RECNUMKSFILE SUBFILE
 *
C           CSTKEY    KLIST
C                     KFLD           ZIP
 ****************************************************
 *       MAINLINE
 ****************************************************
C                     WRITEFOOT1
C                     WRITEHEAD
C                     EXFMTPROMPT
 *
C           *IN03     DOWEQ'0'
C           CSTKEY    SETLLCUSREC                    20
C           *IN20     IFEQ *ZERO
C                     MOVE '1'       *IN61
C                     ELSE
C                     EXSR SFLPRC
C                     END
C           *IN03     IFEQ '0'
C           *IN04     IFEQ '0'
C           *IN61     IFEQ '0'
C                     WRITEFOOT1
C                     WRITEHEAD
C                     END
C                     EXFMTPROMPT
C                     END
C                     END
C                     END
 *
C                     SETON                      LR
```

Figure 9-27 (Part 1 of 2). File Description Specification and Calculation Specification for WORKSTN Subfile Processing Program 1

```
 ********************************************************************
 *    SUBROUTINE - SFLPRC                                           **
 *    PURPOSE    - PROCESS SUBFILE AND DISPLAY                      **
 ********************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
C           SFLPRC    BEGSR
C           NXTPAG    TAG
C                     EXSR SFLCLR
C                     EXSR SFLFIL
C           SAMPAG    TAG
C                     WRITEFOOT2
C                     WRITEHEAD
C                     EXFMTSUBCTL
C           *IN95     IFEQ '1'
C           *IN71     IFEQ '0'
C                     GOTO NXTPAG
C                     ELSE
C                     GOTO SAMPAG
C                     END
C                     END
C                     ENDSR
 ********************************************************************
 *    SUBROUTINE - SFLFIL                                           **
 *    PURPOSE    - FILL SUBFILE                                     **
 ********************************************************************
C           SFLFIL    BEGSR
C           *IN21     DOWEQ'0'
C           ZIP       READECUSREC                   71
C           *IN71     IFEQ '1'
C                     MOVE '1'       *IN21
C                     ELSE
C                     ADD  1         RECNUM
C                     WRITESUBFILE                  21
C                     END
C                     END
C                     ENDSR
 ********************************************************************
 *    SUBROUTINE - SFLCLR                                           **
 *    PURPOSE    - CLEAR SUBFILE RECORDS                            **
 ********************************************************************
C           SFLCLR    BEGSR
C                     MOVE '1'       *IN55
C                     WRITESUBCTL
C                     MOVE '0'       *IN55
C                     MOVE '0'       *IN21
C                     Z-ADD*ZERO     RECNUM  50
C                     ENDSR
```

Figure 9-27 (Part 2 of 2). File Description Specification and Calculation Specification for WORKSTN Subfile Processing Program 1

The file description specifications identify the disk file to be searched and the display device file to be used (CUSSRC). The continuation line for the WORKSTN file identifies the record format (SUBFILE) that is to be used as a subfile. The relative record

number field (RECNUM) specified in positions 47 through 52 of the continuation line controls which record within the subfile is being accessed.

The program displays the PROMPT record format and waits for the work-station user's response. Function key 3 sets on indicator 03, which controls the end of the program. The zip code (ZIP) is used to position the CUSZIPL file by the SETLL operation. Notice that the record format name (CUSREC) is used in the SETLL operation instead of the file name (CUSZIPL). If no record is found, an error message is displayed.

The SFLPRC subroutine handles the processing for the subfile: clearing, filling, and displaying. The subfile is prepared for additional requests in subroutine SFLCLR. If indicator 55 is on, no action occurs on the display, but the main storage area for the subfile records is cleared. The SFLFIL routine fills the subfile with records. A record is read from the CUSZIPL file. If the zip code is the same, the record count (RECNUM) is incremented and the record is written to the subfile. This subroutine is repeated until either the subfile is full (indicator 21 on the WRITE operation) or end of file occurs on the CUSZIPL file (indicator 71 on the READE operation). When the subfile is full or end of file occurs, the subfile is written to the display by the EXFMT operation by the subfile control record control format. The user reviews the display and decides:

- To end the program by pressing function key 3.
- To restart the zip code by pressing function key 4. The PROMPT record format is not displayed, and the subfile is displayed starting over with the same zip code.
- To fill another page by pressing the ROLL UP keys. If end of file has occurred on the CUSZIPL file, the current page is redisplayed; otherwise, the subfile is cleared and the next page is displayed.
- To continue with another zip code by pressing ENTER key. The PROMPT record format is displayed. The user can enter a zip code or end the program.

```
10:11:08              CUSTOMER FILE SEARCH                    01/01/88

Enter Zip Code 72901

  ENTER - Continue      F3 - End Job
```

Figure 9-28. Prompt Screen for WORKSTN Subfile Processing Program

The user enters a zip code in response to the prompt.

```
10:11:23              CUSTOMER FILE SEARCH                    01/01/88

Zip Code  72901

Customer Name         A/R Balance

BRADFIELD             11,111,111.00
LEUNG                 22,222,222.00
ALLEN                 33,333,333.00
BELL                  44,444,444.00
KETCHUM               55,555,555.00
FRASER                66,666,666.00
GOODING               77,777,777.00
LANE                  88,888,888.00
MARSHALL              11,111,111.00
ROBERTS               11,111,222.00
EWING                 33,333,333.00
LOGAN                 44,444,444.00
KENT                  55,555,555.00

  ENTER - Continue      F3 - End Job      F4 - RESTART ZIP CODE
```

Figure 9-29. Display Screen for WORKSTN Subfile Processing Program

The subfile is written to the screen.

Sample Program 5—Inquiry by Zip Code and Search on Name

The following figures illustrate a simple inquiry program using the WORKSTN file:

Table 9-5. List of figures for WORKSTN Inquiry Program	
Figure	**Contents**
Figure 9-30 and Figure 9-31 on page 9-49	DDS specifications for master file and display device file
Figure 9-32 on page 9-52	File Description and Calculation specifications
Figure 9-33 on page 9-55	Prompt screen
Figure 9-34 on page 9-56	Information screen
Figure 9-35 on page 9-56	Detailed information screen

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A* CUSTOMER MASTER FILE -- CUSMSTP
A          R CUSREC
A            CUST           5          TEXT('CUSTOMER NUMBER')
A            NAME          20          TEXT('CUSTOMER NAME')
A            ADDR          20          TEXT('CUSTOMER ADDRESS')
A            CITY          20          TEXT('CUSTOMER CITY')
A            STATE          2          TEXT('CUSTOMER STATE')
A            ZIP            5  0       TEXT('CUSTOMER ZIP CODE')
A            SRHCOD         3          TEXT('CUSTOMER NAME SEARCH CODE')
A            CUSTYP         1          TEXT('CUSTOMER TYPE')
A            ARBAL         10  2       TEXT('ACCOUNTS RECEIVABLE BALANCE')


A*  FILE NAME:   MLGMSTL1
A*  DESCRIPTION: LOGICAL VIEW OF CUSTOMER MASTER FILE (CUSMSTP)
A*               BY ZIP CODE (ZIP) AND NAME(NAME)
A          R CUSREC                    PFILE(CUSMSTP)
A          K ZIP
A          K NAME
```

Figure 9-30. DDS Specifications for Inquiry by Zip Code Master File

The DDS for the data base file used in this program defines one record format named CUSREC, and identifies the ZIP and NAME fields as the key fields.

```
A*  FILE       : MLG265D
A*  DESCRIPTION: DISPLAY CUSTOMER MASTER BY ZIP CODE & NAME
A*****************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A                                     DSPSIZ(24 80 *DS3)
A                                     REF(CUSMSTP)
A                                     CHGINPDFT(CS)
A                                     PRINT(QSYSPRT)
A                                     INDARA
A                                     CA03(03 'END OF JOB')
A          R HEAD
A                                     OVERLAY
A                                   2  4TIME
A                                     DSPATR(HI)
A                                   2 29'Customer Master Inquiry'
A                                     DSPATR(HI UL)
A                                   2 70DATE
A                                     EDTCDE(Y)
A                                     DSPATR(HI)
A          R FOOT1
A                                  23  6'ENTER - Continue'
A                                     DSPATR(HI)
A                                  23 29'F3 - End Job'
A                                     DSPATR(HI)
A          R FOOT2
A                                  23  6'ENTER - Continue'
A                                     DSPATR(HI)
A                                  23 29'F3 - End Job'
A                                     DSPATR(HI)
A                                  23 47'F4 - Restart Zip Code'
A                                     DSPATR(HI)
A          R PROMPT
A                                     OVERLAY
A                                   4  4'Enter Zip Code'
A                                     DSPATR(HI)
A            ZIPCD     R     Y  I  4 19REFFLD(ZIP CUSMSTP)
A                                     CHECK(RZ) DSPATR(CS)
A                                   5  7'Search Name'
A                                     DSPATR(HI)
A            SRCNAM    R        I  5 19REFFLD(NAME CUSMSTP)
A                                     DSPATR(CS)
```

Figure 9-31 (Part 1 of 2). DDS Specifications for Inquiry by Zip Code Display Device File

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A          R SUBFILE
A                                        CHANGE(99 'FIELD CHANGED')
A                                        SFL
A            SEL             1   B  9  8DSPATR(CS)
A                                        VALUES(' ' 'X')
A            ZIP        R        0  9 17
A            CUST       R        0  9 30
A            NAME       R        0  9 43
A          R SUBCTL                      SFLCTL(SUBFILE)
A                                        SFLSIZ(0013)
A                                        SFLPAG(0013)
A  55                                    SFLCLR
A N55                                    SFLDSPCTL
A N55                                    SFLDSP
A                                        ROLLUP(95 'ROLL UP')
A                                        OVERLAY
A                                        CA04(04 'RESTART ZIP CDE')
A                                     4  4'Zip Code'
A            ZIPCD      R        0  4 17REFFLD(ZIP CUSMSTP)
A                                        DSPATR(HI)
A                                     5  4'Search Name'
A            SRCNAM     R        0  5 17REFFLD(NAME CUSMSTP)
A                                        DSPATR(HI)
A                                     7  6'Select'
A                                        DSPATR(HI)
A                                     8  6' "X"       Zip Code     Number   -
A                                           Customer Name       '
A                                        DSPATR(HI)
A                                        DSPATR(UL)
A          R CUSDSP
A                                        OVERLAY
A                                        CA04(04 'RESTART ZIP CDE')
A                                     6 25'Customer'
A            CUST            5A  0  6 35DSPATR(HI)
A                                     8 25'Name'
A            NAME           20A  0  8 35DSPATR(HI)
A                                    10 25'Address'
A            ADDR           20A  0 10 35DSPATR(HI)
A                                    12 25'City'
A            CITY           20A  0 12 35DSPATR(HI)
A                                    14 25'State'
A            STATE           2A  0 14 35DSPATR(HI)
A                                    14 41'Zip Code'
A            ZIP             5S 00 14 50DSPATR(HI)
A                                    16 25'A/R Balance'
A            ARBAL          10Y 20 16 42DSPATR(HI)
A                                       EDTCDE(J)
```

Figure 9-31 (Part 2 of 2). DDS Specifications for Inquiry by Zip Code Display Device File

The DDS for the CUSSRC display device file contains seven record formats: HEAD, FOOT1, FOOT2, PROMPT, SUBFILE, SUBCTL, and CUSDSP.

The PROMPT record format requests the user to enter a zip code and search name. If no entry is made, the display starts at the beginning of the file. The user can press function key 3, which sets on indicator 03, to end the program.

The SUBFILE record format must be defined immediately preceding the subfile control record format (SUBCTL). The subfile record format defined with the keyword SFL, describes each field in the record, and specifies the location where the first record is to appear on the display (here, on line 9).

The subfile control record format (SUBCTL) contains the following unique keywords:

- SFLCTL identifies this format as the control record format and names the associated subfile record format.
- SFLCLR describes when the subfile is to be cleared of existing records (when indicator 55 is on). This keyword is needed for additional displays.
- SFLDSPCTL indicates when to display the subfile control record format (when indicator 55 is off).
- SFLDSP indicates when to display the subfile (when indicator 55 is off).
- SFLSIZ specifies the total size of the subfile. In this example, the subfile size is 15 records that are displayed on lines 9 through 23.
- SFLPAG defines the number of records on a page. In this example, the page size is the same as the subfile size.
- ROLLUP indicates that indicator 95 is set on in the program when the roll up function is used.

The OVERLAY keyword defines this subfile control record format as an overlay format. This record format can be written without OS/400 erasing the screen first. Function key 3 is valid for repeating the search with the same zip code. (This use of function key 3 allows a form of rolldown.)

The CUSDSP record format displays information for the selected customers.

```
*****************************************************************
*    PROGRAM ID   - MLG265                                      **
*    PROGRAM NAME - MAILING LIST SEARC BY ZIP CODE/NAME         **
*****************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
FMLGMSTL1IF  E           K       DISK
FMLG265D CF  E                   WORKSTN
F                                      RECNUMKSFILE SUBFILE
C           CSTKEY    KLIST
C                     KFLD           ZIPCD
C                     KFLD           SRCNAM
C           ZIPKEY    KLIST
C                     KFLD           ZIP
C                     KFLD           NAME
*****************************************************************
*       MAINLINE                                                **
*****************************************************************
C                     WRITEFOOT1
C                     WRITEHEAD
C                     EXFMTPROMPT
*
C           *IN01     DOWEQ'0'
C           CSTKEY    SETLLCUSREC
C                     EXSR SFLPRC
C                     EXSR SFLCHG
C           *IN03     IFEQ '0'
C           *IN04     ANDEQ'0'
C                     WRITEFOOT1
C                     WRITEHEAD
C                     EXFMTPROMPT
C                     END
C                     END
*
C                     SETON                     LR
```

Figure 9-32 (Part 1 of 3). File Description Specification and Calculation Specification for Inquiry by Zip Code and Search on Name Program

```
 ****************************************************************
 *    SUBROUTINE - SFLPRC                                       **
 *    PURPOSE    - PROCESS SUBFILE AND DISPLAY                  **
 ****************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
C           SFLPRC    BEGSR
C           NXTPAG    TAG
C                     EXSR SFLCLR
C                     EXSR SFLFIL
C           SAMPAG    TAG
C                     WRITEFOOT2
C                     WRITEHEAD
C                     EXFMTSUBCTL
C           *IN95     IFEQ '1'
C           *IN71     IFEQ '0'
C                     GOTO NXTPAG
C                     ELSE
C                     GOTO SAMPAG
C                     END
C                     END
C                     ENDSR
 ****************************************************************
 *    SUBROUTINE - SFLFIL                                       **
 *    PURPOSE    - FILL SUBFILE                                 **
 ****************************************************************
C           SFLFIL    BEGSR
C           *IN21     DOWEQ'0'
C                     READ CUSREC                   71
C           *IN71     IFEQ '1'
C                     MOVE '1'       *IN21
C                     ELSE
C                     ADD  1         RECNUM
C                     MOVE *BLANK    SEL
C                     WRITESUBFILE                  21
C                     END
C                     END
C                     ENDSR
```

Figure 9-32 (Part 2 of 3). File Description Specification and Calculation Specification for Inquiry by Zip Code and Search on Name Program

```
*****************************************************************
*    SUBROUTINE - SFLCLR                                        **
*    PURPOSE    - CLEAR SUBFILE RECORDS                         **
*****************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
C           SFLCLR    BEGSR
C                     MOVE '1'       *IN55
C                     WRITESUBCTL
C                     MOVE '0'       *IN55
C                     MOVE '0'       *IN21
C                     Z-ADD*ZERO     RECNUM  50
C                     ENDSR
*****************************************************************
*    SUBROUTINE - SFLCHG                                        **
*    PURPOSE    - CUSTOMER RECORD SELECTED                      **
*****************************************************************
C           SFLCHG    BEGSR
C                     READCSUBFILE                  98
C           *IN98     IFEQ '0'
C           ZIPKEY    CHAINCUSREC              71
C                     EXFMTCUSDSP
C                     END
C                     ENDSR
```

Figure 9-32 (Part 3 of 3). File Description Specification and Calculation Specification for Inquiry by Zip Code and Search on Name Program

The file description specifications identify the disk file to be searched and the display device file to be used (MLG265D). The continuation line for the WORKSTN file identifies the record format (SUBFILE) to be used as a subfile. The relative record number field (RECNUM) specified in positions 47 through 52 of the continuation line controls which record within the subfile is being accessed.

The program displays the PROMPT record format and waits for the work-station user's response. Function key 3 sets on indicator 03, which controls the end of the program. The zip code (ZIP) and name (NAME) are used as the key to position the MLGMSTL1 file by the SETLL operation. Notice that the record format name (CUSREC) is used in the SETLL operation instead of the file name (MLGMSTL1).

The SFLPRC subroutine handles the processing for the subfile: clearing, filling, and displaying. The subfile is prepared for additional requests in subroutine SFLCLR. If indicator 55 is on, no action occurs on the display, but the main storage area for the subfile records is cleared. The SFLFIL routine fills the subfile with records. A record is read from the MLGMSTL1 file, the record count (RECNUM) is incremented, and the record is written to the subfile. This subroutine is repeated until either the subfile is full (indicator 21 on the WRITE operation) or end of file occurs on the MLGMSTL1 file (indicator 71 on the READ operation). When the subfile is full or end of file occurs, the subfile is written to the display by the EXFMT operation by the subfile control record control format. The user reviews the display and decides:

- To end the program by pressing function key 3.
- To restart the subfile by pressing function key 4. The PROMPT record format is not displayed, and the subfile is displayed starting over with the same zip code.

- To fill another page by pressing the ROLL UP keys. If end of file has occurred on the MLGMST1 file, the current page is redisplayed; otherwise, the subfile is cleared, and the next page is displayed.
- To display customer detail by entering X, and pressing ENTER. The user can then return to the PROMPT screen by pressing ENTER, redisplay the subfile by pressing function key 4, or end the program by pressing function key 3.

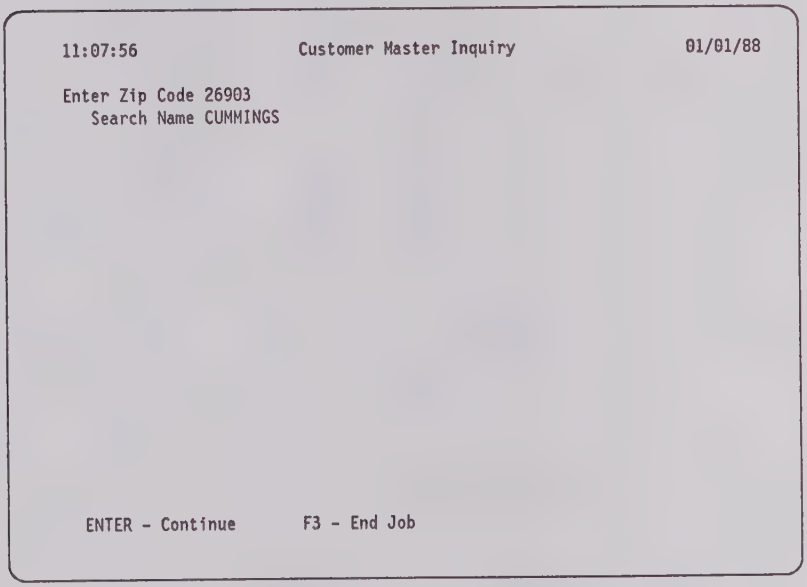


Figure 9-33. Prompt Screen for Zip Code Search

The user responds to the initial prompt by entering a Zip Code and name.

```
11:09:20              Customer Master Inquiry                01/01/88

Zip Code     26903
Search Name  CUMMINGS

  Select
   "X"       Zip Code     Number       Customer Name
             26903        00011        CUMMINGS
             26903        00012        DONLEAVY
             26903        00013        DREYFUS
             26903        00014        FREDERICKS
             26903        00015        RYERSON
     X       26903        00016        SANDFORD
             26903        00017        STEVENS
             26903        00018        TALLBOY
             26903        00019        TORRENCE
             26903        00020        WALTERS
             27810        00021        GRAY
             27810        00022        GRAYSON
             27810        00023        HALIBURTON

  ENTER - Continue      F3 - End Job      F4 - Restart Zip Code
```

Figure 9-34. Information Display for Zip Code Search

The user requests more information by entering "X".

```
11:09:20              Customer Master Inquiry                10/23/87

                    Customer  00016

                    Name      SANDFORD

                    Address   40 YONGE EAST

                    City      HAMILTON

                    State     WA    Zip Code 26903

                    A/R Balance            100.00




  ENTER - Continue      F3 - End Job      F4 - Restart Zip Code
```

Figure 9-35. Detailed Information Display for Zip Code Search

The user selects the appropriate function key to continue or end the inquiry.

Sample Program 6—Program Described WORKSTN File within FORMAT Name on Output Specifications

The following figures illustrate the use of a WORKSTN within FORMAT name on Output Specifications.

Table 9-6. List of figures for FORMAT Name on Output Specifications	
Figure	**Contents**
Figure 9-36	DDS specifications for display device file
Figure 9-37 on page 9-58	File Description, Input, Calculation and Output Specifications

```
 *...1....+....2....+....3....+....4....+....5....+....6....+....7...*
A*  FILE       : CUSINQ
A*  DESCRIPTION: DISPLAY FILE FOR FORMAT NAME ON OUTPUT
A          R ITMPMT
A                                        TEXT('INVENTORY INQUIRY PROMPT')
A                                        CF01(15 'END OF PROGRAM')
A                                    1  2'Inventory Inquiry Prompt'
A                                    2  2'Enter Item Number'
A            RECID          1   I  2 23DFT('A') DSPATR(ND PR)
A            ITEM           5   I  2 25
A  99                                    ERRMSG('Item Not Found' 99)
A          R ITMDTL                      TEXT('INVENTORY DETAIL')
A                                        OVERLAY
A                                    5  2'Item No.'
A                                    5 14'Description'
A                                    5 41'Price'
A                                    5 53'Sold'
A                                    5 62'Onhand'
A            ITEM           5        7  2
A            DESCRP        20        7 14
A            PRICE          8        7 41
A            PENDNG         5        7 53
A            ONHAND         5        7 62
```

Figure 9-36. DDS Specifications for Program Described WORKSTN File within FORMAT Name on Output Specifications

The data description specifications for the display device file CUSINQ describe how the data sent from RPG/400 is displayed on the screen.

```
 *...1....+....2....+....3....+....4....+....5....+....6....+....7...*
FCUSINQ  CP  F     50           WORKSTN      KPASS  *NOIND
FINVMSTL IF  E          K       DISK
 *
ICUSTINQ NS  03   2 CA
I                                       1   1 *IN15
I                                       2   2 RECID
I                                       3   7 ITEM
 *
C           *IN03     IFEQ '1'
C           ITEM      CHAININVDTL              99
C                     END
C           *IN15     IFEQ '1'
C                     MOVE '1'       *INLR
C                     RETRN
C                     END
 *
OCUSINQ  D        1P
O       OR        03
O       OR        99
O                                  K6 'ITMPMT'
O                           *IN99   1
O        D        03N99
O                                  K6 'ITMDTL'
O                           ITEM    5
O                           DESCRP 25
O                           PRICE  33
O                           PENDNG 38
O                           ONHAND 43
```

Figure 9-37. File Description, Input, Calculation, and Output Specifications for Program Described WORKSTN File within FORMAT Name on Output Specifications

On the output specifications, because the format name ITMPMT is conditioned by 1P, it is written to the file before any input operations take place. This format is also written to the file when indicator 03 or indicator 99 is on. If indicator 99 is on, the error message that is defined in DDS is displayed. To pass indicator 99 on output, the field *IN99 is defined in the output record. The format ITMDTL is written to the file when indicator 03 is on and indicator 99 is not on. The end positions for the fields must be the same as the end positions defined on the DDS listing.

Sample Program 7—Variable Start Line

```
     *...1....+....2....+....3....+....4....+....5....+....6....+....7...*
    A*  FILE       : INQUIRY
    A*  DESCRIPTION: DISPLAY FILE FOR VARIABLE START LINE
    A                                     PRINT
    A          R PROMPT                   SLNO(*VAR)
    A            MONTH          9A  0  6 15DSPATR(HI)
    A            DAY            2   0  6 26DSPATR(HI)
    A            YR             2   0  6 30DSPATR(HI)
    A                                   6 45TIME   DSPATR(HI)
     ************************************************************************
     *                                                                      **
     *    PROGRAM ID   - VARLINE                                            **
     *    PROGRAM NAME - VARIABLE START LINE DISPLAY                        **
     *                                                                      **
     ************************************************************************
    FINQUIRY CF  E                    WORKSTN
    F                                             KSLN   SLNFLD
    E                   TABM    1  12  2 0 TABD    9    TABLE OF MONTHS
     *
    C                    MOVE UDAY      DAY
    C                    MOVE UYEAR     YR
    C           UMONTH   LOKUPTABM      TABD           66
    C           *IN66    IFEQ '1'
    C                    MOVE TABD      MONTH
    C                    END
    C                    Z-ADD6         SLNFLD  20
     *
    C                    EXFMTPROMPT
     *
    C                    MOVE '1'       *INLR
**
01JANUARY
02FEBRUARY
03MARCH
04APRIL
05MAY
06JUNE
07JULY
08AUGUST
09SEPTEMBER
10OCTOBER
11NOVEMBER
12DECEMBER
```

Figure 9-38. Variable Start Line

A start line number (SLN) field determines the line number where a record format is written to a display file. SLN can be specified for both program described and externally described files. To use a variable start line for a display file record format, specify the SLN option on the file continuation specifications. The DDS for the file must specify SLNO(*VAR) for one or more record formats. Only these record formats are affected by the value of the SLN field.

On output operations to the file, the value of the SLN field determines the line number where record formats are actually written. If the SLN field has a value of 1 through 24, 1 is subtracted from the value, and the result is added to the line numbers specified in the DDS. The resulting values are used as the actual line numbers for writing the fields and constants specified in the DDS. However, the start line for the record format is the value of the SLN field. This means that the record format written occupies all the lines between the start of the format and the highest actual line number written to the display. If the SLN field has a value of 0, a format appears on the display as if an SLN field value of 1 were specified. If the value of the SLN field is negative or greater than 24, an RPG/400 1299 error message is issued.

For more information, see the *Programming: Data Management Guide* and the *Programming: Data Description Specifications Reference.*

In this example , the EXFMT operation uses a start line number field (SLNFLD) with a value of 6. This causes the record format to be displayed starting at line 06, the output fields are written to line 11:

```
(6(SLNFLD) - 1 + 6(DDS start line number)).
```

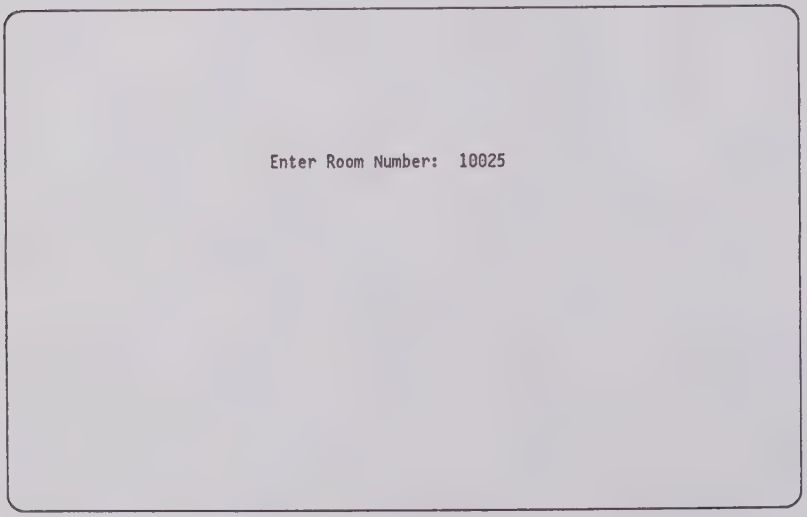


Figure 9-39. Prompt Screen for Variable Start Line

Sample Program 8—Read Operation with Time-Out

```
     *...1....+....2....+....3....+....4....+....5....+....6....+....7...*
    A*  FILE       : HOTELDSP
    A*  DESCRIPTION: DISPLAY FILE FOR TIME OUT EXAMPLE
    A                                     INVITE
    A          R REQUEST
    A                                     OVERLAY
    A            ROOM           5A  I 10 46DSPATR(HI)
    A                                  10 26'Enter Room Number:'
    A                                     DSPATR(HI)
```

Figure 9-40. DDS Specifications for Read Operation with Time-Out

```
     ******************************************************************
     *                                                                **
     *    PROGRAM ID   - TIMEOUT                                      **
     *    PROGRAM NAME - TIME OUT ON READ                             **
     *                                                                **
     ******************************************************************
    FHOTELDSPCF  E                    WORKSTN
    F                                           KNUM        1
    F                                           KINFDS FEEDBK
    IFEEDBK      DS
    I                                    *STATUS  STATUS
    .

    .

    .

     *
    C                    WRITEREQUEST
    C                    READ HOTELDSP                9950
    C                    EXSR ERRCHK
    C                    MOVE '1'       *INLR
     ******************************************************************
     *    SUBROUTINE - ERRCHK                                         **
     *    PURPOSE    - CHECK STATUS FOR MAX WAIT                      **
     ******************************************************************
    C           ERRCHK   BEGSR
    C           STATUS   IFEQ 1331
    C                    MOVE 'SIGNOFF' CMD     7
    C                    Z-ADD7         LEN    155
    C                    CALL 'QCMDEXC'
    C                    PARM           CMD
    C                    PARM           LEN
    C                    END
    C                    ENDSR
```

Figure 9-41. File Description, Input, and Calculation Specifications for Read Operation with Time -Out

This program causes the work station to be signed off, when no work-station activity has occurred during a specified length of time.

- In the DDS for the display file HOTELDSP, the keyword INVITE is specified for all formats. You specify a length of time to wait with the WAITRCD parameter on the CRTDSPF (or CHGDSPF) command to create (or change) this file.
- In the file specifications, the file HOTELDSP is specified as a WORKSTN file with the option NUM. RPG treats the file as a multiple device file.
- In the input specifications, the *STATUS subfield of the file information data structure is named STATUS.
- The WRITE operation puts format REQUEST on the work station and, because of the INVITE keyword, makes the work station an invited device.
- The READ-by-file-name operation to the file HOTELDSP will wait the length of time specified on the WAITRCD parameter for a response from the invited device.
- If no response comes in time, error indicator 99 is set on and the program continues with the next operation.
- The next operation performs subroutine ERRCHK. This subroutine checks the STATUS subfield of the file information data structure. Status code 1331 indicates the READ operation timed out, and the ERRCHK subroutine signs the work station off. Other status codes produce other results.

Note: This example is not a complete program.

Chapter 10. Data Field Formats and Data Structures

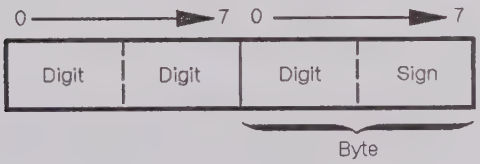
RPG/400 works with data that is stored in fields in data files. Within these files, the fields can be grouped together into data structures.

Format of Fields in Files

RPG/400 input and output fields can be in character, zoned decimal, packed decimal, or binary format. A leading or trailing sign can be specified only with zoned decimal format. All numeric input fields (unless they are in a data structure) are converted by the compiler to packed decimal format for internal processing. The program runs in the same way whether numeric data is in packed decimal format, zoned decimal format, or binary format. However, the system processes arithmetic calculations more efficiently if the data is in packed decimal format. Subfields within a data structure are always carried in the format specified by the subfield specification.

Packed Decimal Format

Packed decimal format means that each byte of storage (except for the low-order byte) can contain two decimal numbers. Each byte (except the low-order byte) is divided into two 4-bit digit portions. The low-order byte contains one digit in the leftmost portion and the sign (+ or -) in the rightmost portion. The standard signs are used: hexadecimal F for positive numbers and hexadecimal D for negative numbers. The packed decimal format looks like this:



The sign portion of the low-order byte indicates whether the numeric value represented in the digit portions is positive or negative. Figure 10-1 on page 10-5 shows what the decimal number 8191 looks like in packed decimal format.

For a program described file, you specify packed decimal input, output, and array or table fields with the following entries:

Packed decimal input field: Specify P in position 43 of the input specifications.

Packed decimal output field: Specify P in position 44 of the output specifications. This position must be blank if editing is specified.

Packed decimal array or table field: Specify P in position 43 and/or position 55 of the extension specifications. Arrays and tables loaded at compile time cannot be in packed decimal format.

For an externally described file, the data format is specified in position 35 of the data description specifications.

Use the following formula to find the length in digits of a packed decimal field:

Number of digits = 2n - 1,

where n = number of packed input record positions used.

This formula gives you the maximum number of bytes you can represent in packed decimal format; the upper limit is 30.

Table 10-1. Packed Equivalents for Zoned Decimal Fields up to 16 Digits Long

Zoned Decimal Length in Digits	Number of Bytes Used in Packed
1	1
3	2
5	3
. . .	. . .
29	15
30	16

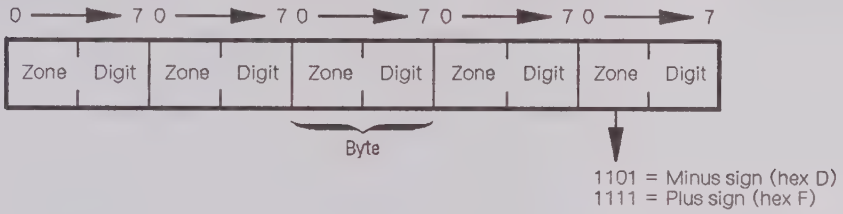
For example: An input field read in packed decimal format has a length of five positions (as specified on the input or data description specifications). The number of digits in this field equals 2(5)-1 or 9. Therefore, when the field is used in the calculation specifications, the result field must be nine positions long.

When a packed decimal field in one program is converted to a zoned decimal field in another program, the zoned decimal field always contains an odd number of bytes. If a field is in packed decimal format in one program and then is unpacked in another program, the field length can increase by 1. If a field is packed and then unpacked in the same program, the field length does not change. This must be considered when fields are packed for storage on an intermediate device and then used by another program.

Packed fields can be up to 16 bytes long. The chart in Table 10-1 shows the packed equivalents for zoned decimal fields up to 16 digits long:

Zoned Decimal Format

Zoned decimal format means that each byte of storage can contain one digit or one character. Any character or numeric field can be read in zoned decimal format. In the zoned decimal format, each byte of storage is divided into two portions: a 4-bit zone portion and a 4-bit digit portion. The zoned decimal format looks like this:



The zone portion of the low-order byte indicates the sign (+ or -) of the decimal number. The standard signs are used: hexadecimal F for positive numbers and hexadecimal D for negative numbers. In zoned decimal format, each digit in a decimal number includes a zone portion; however, only the low-order zone portion serves as the sign. Figure 10-1 on page 10-5 shows what the decimal number 8191 looks like in zoned decimal format.

You must also consider the change in field length when coding the end position in positions 40 through 43 of the output specifications. To find the length of the field after it has been packed, use the following formula:

$$\text{Field length} = \frac{N}{2} + 1$$

...where n = number of digits in the zoned decimal field.

(Any remainder from the division is ignored.)

For a program described file, zoned decimal format is specified by a blank in position 43 of the input specifications, in position 44 of the output specifications, or in position 43 or 55 of the extension specifications. For an externally described file, the data format is specified in position 35 of the data description specifications.

Binary Format

Binary format means that the sign (+ or -) is in the leftmost bit of the field and the integer value is in the remaining bits of the field. Positive numbers have a zero in the sign bit; negative numbers have a one in the sign bit and are in twos complement form. In binary format, each field must be either 2 or 4 bytes long.

Every input field read in binary format is assigned a field length (number of digits) by the compiler. A length of four is assigned to a 2-byte binary field; a length of nine is assigned to a 4-byte binary field. Because of these length restrictions, the highest decimal value that can be assigned to a 2-byte binary field is 9999 and the highest decimal value that can be assigned to a 4-byte binary field is 999 999 999.

For a program described file, specify binary input, binary output, and binary array or table fields with the following entries:

- *Binary input field:* Specify B in position 43 of the input specifications.
- *Binary output field:* Specify B in position 44 of the output specifications. This position must be blank if editing is specified.

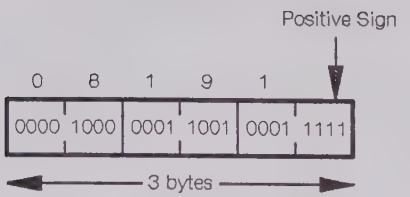
  The length of a field to be written in binary format cannot exceed nine digits. If the length of the field is from one to four digits, the compiler assumes a binary field length of two bytes. If the length of the field is from five to nine digits, the compiler assumes a binary field length of 4 bytes.

  Since a 2-byte input field in binary format is converted by the compiler to a 4-digit decimal field, it is possible for the input value to be too large. If this happens, the leftmost digit of the number is dropped. For example, an input field has a binary value of hex 7000. The compiler converts this to 28 672 in decimal. The 2 is dropped and the result is 8672.
- *Binary array or table field:* Specify B in position 43 and/or position 55 of the extension specifications. Arrays and tables loaded at compile time cannot be in binary format.

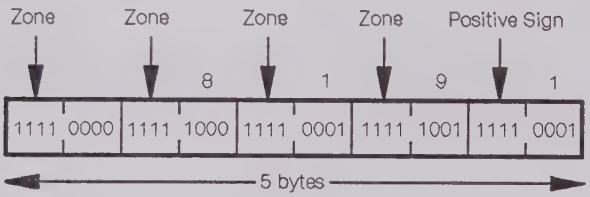
For an externally described file, the data format is specified in position 35 of the data description specifications.

Figure 10-1 on page 10-5 shows what the decimal number 8191 looks like in various formats.

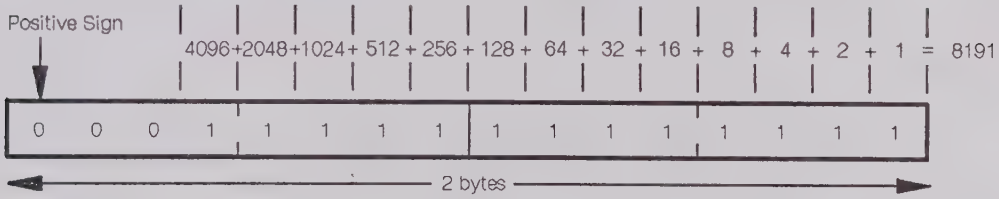
Packed Decimal Format:



Zoned Decimal Format: [1]



Binary Format: [2]



[1]If 8191 is read into storage as a zoned decimal field, it occupies 4 bytes. However, if it is converted to packed decimal format, it occupies 3 bytes; then when it is converted back to zoned decimal format, it occupies 5 bytes.

[2]To obtain the numeric value of a positive binary number and the values of the bits that are on (1); the sign bit is not included. To obtain the numeric value of a negative binary number, add the values of the bits that are off (0) plus one; the sign bit is not included.

Figure 10-1. Binary, Packed, and Zoned Decimal Representation of the Number 8191

Signs

RPG/400 ensures that a consistent plus or minus sign is present for all numeric fields. The standard signs for all packed and zoned numeric fields are hexadecimal F for plus and hexadecimal D for minus.

External Formats

When a sign is written out for numeric fields, the sign (+ or -) is included in the units position of the data field unless editing has been done (see Chapter 14, "Editing Numeric Fields").

You can specify an alternative sign format for zoned decimal format. In the alternative sign format the numeric field is immediately preceded or followed by a + or - sign. A plus sign is a hexadecimal 4E and a minus sign is a hexadecimal 60.

For program described files, specify preceding (L entry) or following (R entry) plus or minus signs in the following positions:

Input field:	Position 43 of the input specifications
Output field:	Position 44 of the output specifications
Array or table field:	Position 43 and/or position 55 of the extension specifications.

When an alternative sign format is specified, the field length must include an additional position for the sign. For example, if a field is 5 digits in length and the alternative sign format is specified, a field length of 6 positions must be specified.

Internal Format

All numeric fields, except subfields of a data structure, are stored in packed decimal format for internal processing. In packed decimal format, the sign is stored in the last four bits of the rightmost byte of the field (see Figure 10-1 on page 10-5).

Data Structures

RPG/400 allows you to define an area in storage and the layout of the fields, called subfields, within the area. This area in storage is called a data structure. You can use a data structure to:

- Define the same internal area multiple times using different data formats
- Operate upon a field and change its contents
- Divide a field into subfields without using the MOVE or MOVEL operation codes
- Define a data structure and its subfields in the same way a record is defined
- Define multiple occurrences of a set of data
- Group noncontiguous data into contiguous internal storage locations.

In addition, there are three special data structures each with a specific purpose:

- A data area data structure (identified by a U in position 18 of the data structure statement)
- A file information data structure (referred to by the keyword INFDS on a file description specifications continuation line)
- A program status data structure (identified by an S in position 18 of the data structure statement).

Data structures can be program described or externally described.

A program described data structure is identified by a blank in position 17 of the data structure statement. The subfield specifications for a program described data structure must immediately follow the data structure statement.

An externally described data structure, identified by an E in position 17 of the data structure statement, has subfield descriptions contained in an externally described file with one record format. At compile time RPG/400 uses the external name to locate and extract the external description of the data structure subfields. An external subfield name can be renamed in the program, and additional subfields can be added to an externally described data structure in the program.

For examples of data structures, see "Data Structure Examples" on page 10-13.

Format of Data Structure Subfields in Storage

Subfields in a data structure are stored in the format specified in position 43 of the data structure subfield specifications. The possible entries for a program described data structure are:

Entry	Explanation
Blank	Subfield is in zoned decimal format or is character data, depending on the entry in position 52 of the subfield specifications.
P	Subfield is in packed decimal format.
B	Subfield is in binary format.

Because the subfields of a data structure are maintained in the format specified, the compiler generates the necessary conversions to process the required function. These conversions can occur at the following times:

- When a record is being read
- At detail or total calculation time
- At detail or total output time.

A data structure is considered to be a character field and is initialized to blanks. Numeric subfields must be initialized with numeric data (with a Z-ADD operation) before they are used in arithmetic or editing operations.

The rules for determining the length of a subfield in packed decimal format, zoned decimal format, and binary format are the same as those for determining the length of a field in packed decimal format, zoned decimal format, and binary format (see "Packed Decimal Format" on page 10-1, "Zoned Decimal Format" on page 10-3, and "Binary Format" on page 10-3).

Data Structure Statement Specifications

Data structure statements are defined on the input specifications and must follow all input specifications for records. The specifications for data structure statements are:

Table 10-2. Specifications For Data Structure Statements

Position	Entry
6	I
7-12	Name of the data structure being defined. This entry is optional for a program described data structure, and is required for an externally described data structure, a file information data structure (INFDS), and a data area data structure.
13-16	Blank
17	*Blank:* Program described data structure. *E:* Externally described data structure. The data structure subfield definitions are retrieved from an externally described record format.
18	*Blank:* Not a program status or data area data structure. *U:* Data area data structure. *S:* Program status data structure.
19-20	DS
21-30	*Blank:* The data structure is program described. *Entry:* This is the name of the file whose first record format contains the field descriptions used as the subfield descriptions for this data structure.
31-43	Blank
44-47	*Blank:* A single occurrence data structure. *nnnn:* A number (right-adjusted) indicating the number of occurrences of the data structure. **Note:** This entry must be blank for a data area data structure, a file information data structure, and a program status data structure.
48-51	Length of data structure (optional). This entry must be right-adjusted.
52-74	Blank

Rules for Specifying Data Structure Statements

Remember the following when you specify data structure statements:

- The data structure name must be a symbolic name with a maximum of 6 characters. The name can appear on only one data structure specification, cannot be a look-ahead field, and can be specified anywhere a character field is allowed.
- All entries for one data structure and its subfields must appear together; they cannot be mixed with entries for other data structures.

- The length of a data structure is one of the following:
  - The length specified in the input field specifications if the data structure name is an input field
  - The highest To position of a subfield within a data structure if the data structure name is not an input field
  - The length specified in positions 48 through 51 of the data structure statement.
- The data structure length is determined by the first specification in the program that defines a length in one of the preceding ways. Subsequent conflicting lengths are invalid.
- A compile-time or run time array cannot be used in a data area data structure or in a multiple occurrence data structure.
- Data structures are character data and can be from 1 to 9999 characters in length.
- A data structure and a subfield of a data structure cannot have the same name.

Multiple Occurrence Data Structure

A multiple occurrence data structure is a data structure whose definition is repeated in a program to form a series of data structures with identical formats. You specify the number of occurrences of a data structure in positions 44 through 47 of the data structure statement. When positions 44 through 47 do not contain an entry, the data structure is not a multiple occurrence data structure. All occurrences of a data structure have the same attributes and can be referred to individually. The OCUR operation code, which can only be used with a multiple occurrence data structure, allows you to specify which occurrence of a data structure is to be used for subsequent operations within the program.

Note: Multiple occurrences are not allowed for a data area, file information, or program status data structure.

For examples for multiple occurrence data structures, see "Data Structure Examples" on page 10-13.

Special Data Structures

Special data structures include:

- Data area data structures
- File information data structures (INFDS)
- Program status data structures.

Data Area Data Structure

A data area data structure, which you identify by a U in position 18 of the data structure statement, indicates to the RPG/400 program that it should read in and lock the data area of the same name at program initialization and should write out and unlock the same data area at the end of program. Data area data structures, as in all other data structures, have the type character. A data area read into a data area data structure must also be character. The data area and data area data structure must

have the same name unless you rename the data area within RPG/400 using the *NAMVAR DEFN statement.

You can specify the data area operations (IN, OUT, and UNLCK) have the type for a data area that is implicitly read in and written out. Before you use a data area data structure with these operations, you must specify that data area in the result field of the *NAMVAR DEFN statement.

A data area data structure cannot be specified in the result field of a PARM operation.

If you specify blanks for the data area data structure (positions 7 through 12 of the input specifications line that contains a U in position 18), RPG/400 uses a local data area. To provide a name for a local data area, use the *NAMVAR DEFN operation, with *LDA in factor 2 and the name in the result field.

For general information on data areas, see Chapter 11, "Communicating with Objects in the System."

File Information Data Structure (INFDS)

You can specify a file information data structure (defined by the keyword INFDS on a file description specifications continuation line) for each file in the program. This provides you with status information on the file exception/error that occurred. The file information data structure name must be unique for each file. A file information data structure contains predefined subfields that provide information on the file exception/error that occurred. For a discussion of file information data structures and their subfields, see "Exception/Error Handling" on page 4-35.

Define and name a file information data structure on a file description specifications continuation line with the following entries:

Table 10-3. Entries to Define and Name a File Information Data Structure

Position	Entry
6	F
7-52	Blank (if the information is specified on a separate continuation line)
53	K (indicates a continuation line)
54-59	INFDS (identifies this data structure as the file information data structure)
60-65	Name of the file information data structure.

Program Status Data Structure

A program status data structure, identified by an S in position 18 of the data structure statement, provides program exception/error information to the program. For a discussion of program status data structures and their predefined subfields, see "Exception/Error Handling" on page 4-35.

Data Structure Subfield Specifications

The subfields of a program described data structure must immediately follow the data structure specification statement to which they apply. The subfields of an externally described data structure are described externally to the RPG/400 program. The subfield specifications are brought into the RPG/400 program at compilation time. The subfields of an externally described data structure can be renamed and/or additional subfield specifications can appear following the data structure statement. All renamed subfields must precede any additional subfield specifications. To add additional subfields to an externally described data structure, follow the same rules as for subfields for a program described data structure. The internally described subfields are added to the retrieved descriptions.

The specifications for subfields are as follows:

Table 10-4 (Page 1 of 2). Specifications for Subfields

Position	Entry
6	I.
7-20	Blank.
21-30	To rename a subfield in an externally described data structure, specify the external name in positions 21 through 30, and specify the name to be used in the program in positions 53 through 58. Leave the other positions blank.
31-42	Blank.
43	*P:* Indicates that the subfield is in packed decimal format. *B:* Indicates that the subfield is in binary format. *Blank:* Indicates that the subfield is in zoned decimal format, or is character data.
44-47 48-51	*1- to 4-digit numbers:* Positions 44 through 47 contain the beginning position, and positions 48 through 51 contain the end position of the subfield. These entries must be right-adjusted; leading zeros can be omitted. or *Keywords:* For a program status data structure or a file information data structure (INFDS), place a special keyword (left-adjusted) in this position. A keyword can start at position 44 and extend through to position 51. See "Exception/Error Handling" on page 4-35 for the keywords and their descriptions.

Table 10-4 (Page 2 of 2). Specifications for Subfields	
Position	**Entry**
52	*0-9:* Indicates the number of decimal positions in a numeric field or an array. *Blank:* Indicates a character field. **Note:** This position must contain an entry for a numeric subfield; however, an entry is not required for an array. If an entry is made for an array, the entry must be the same as that specified in the extension specifications.
53-58	The subfield name. **Note:** If an array is specified as a subfield name, the length indicated in positions 44 through 51 must equal the entire amount of storage required to store the array (for example, 10 binary halfword elements require 20 bytes of storage).
59-74	Blank.

Rules for Subfield Specifications

Remember the following when you specify subfield specifications:

- If the length (positions 44 through 51) or decimal positions (position 52) for the subfield differ from prior definitions in the program, the first definition is used and subsequent conflicting definitions are invalid.
- If the To position (48 through 51) specified for a subfield is larger than the defined length of an input field of the same name or the defined length of the data structure, the subfield specification is invalid.
- To redefine subfields, specify the same or part of the same From and To positions (44 through 51) for another subfield in the same data structure.
- To define a single position subfield, enter the same number in both positions 44 through 47 and positions 48 through 51.
- Overlapping subfields cannot be used in the same calculation specification.
- If an array or array element with a variable index is specified in the calculation specifications in factor 1, factor 2, or the result field, the entire array is used to determine whether overlap exists.
- You must ensure that packed, zoned, or binary numeric subfields are initialized with numeric data before they are used in arithmetic or editing operations.
- An input field name cannot:
  - Appear as both a subfield name and a data structure name
  - Appear more than once as a subfield name.
- The following calculation operations are checked for overlapping subfields:

- Factor 1 and the result field, and factor 2 and the result field of ADD, SUB, MULT, DIV, Z-ADD, and Z-SUB. Factor 1 and factor 2 of the preceding operations may overlap.
- Factor 2 and the result field of a MOVE, MOVEL, or MOVEA operation are checked for overlap.
- Factor 2 and the result field and factor 1 and the result field of a PARM operation are checked for overlap.

Data Structure Examples

Figure 10-2 through Figure 10-8 on page 10-20 show some typical uses for data structures.

I

Line	Form Type	Filename or Record Name	Data Structure Name	Sequence	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, *, or DS	External Field Name / Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location: From	Field Location: To	Data Structure: Occurs n Times	Data Structure: Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Field Indicators: Minus	Field Indicators: Zero or Blank
01	I	FILEIN		NS			Ø1	1		CA		2		CB																			
02	I																						3	18		PARTNO						
03	I																						19	29		NAME						
04	I																						30	40		PATNO						
05	I																						41	61		DR						
06	I	PARTNO					DS																									
07	I																						1	4		MFG						
08	I																						5	10		DRUG						
09	I																						11	13		STRNTH						
10	I																						14	16Ø		COUNT						
11	I																															

The data structure subfields can be referred to by PARTNO name or by subfields MFG, DRUG, STRNTH, or COUNT.

Figure 10-2. Using a Data Structure to Define Subfields within a Field

Data Structure Examples

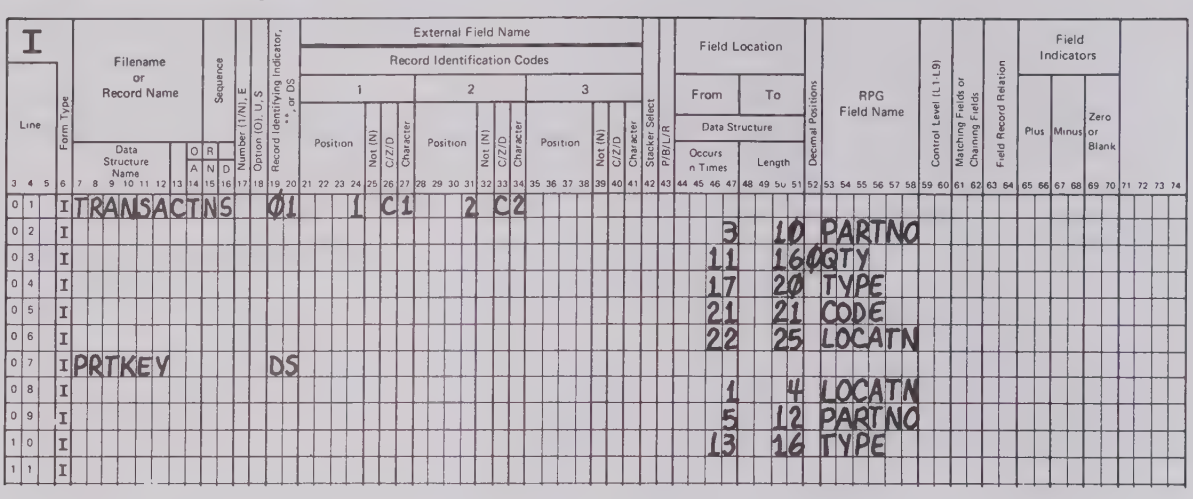
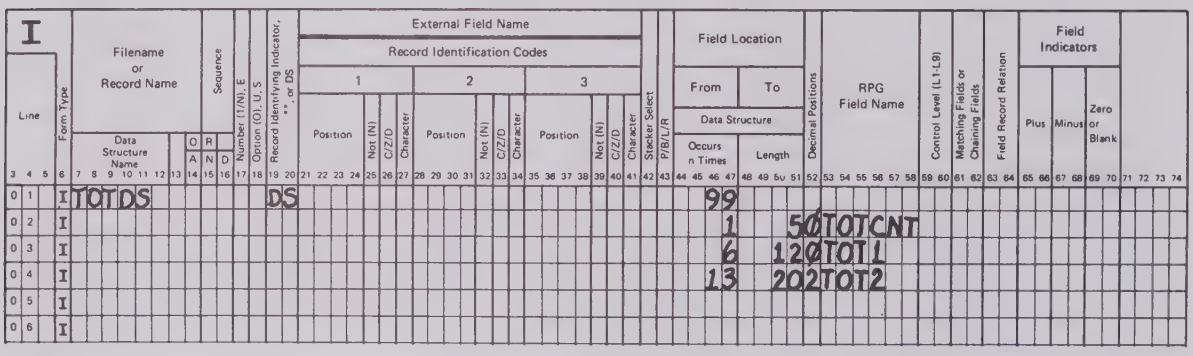


Figure 10-3. Using a Data Structure to Group Fields

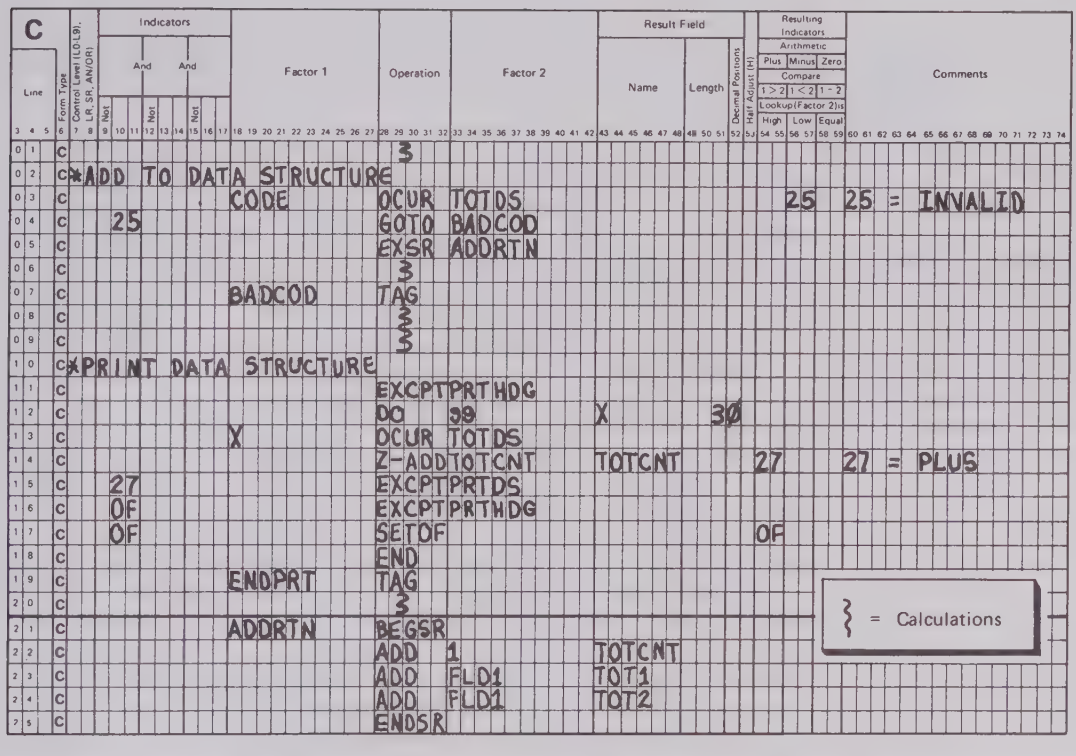
When you use a data structure to group fields, fields from nonadjacent locations on the input record can be made to occupy adjacent internal locations. The area can then be referred to by the data structure name and/or individual subfield name.



In this example, a multiple occurrence data structure is used to accumulate a series of totals for specific codes, and the totals of each of the occurrences of the data structure are written.

The program described data structure, TOTDS, has 99 occurrences (positions 46 and 47). The length of the data structure can be specified in positions 48 through 51.

Figure 10-4 (Part 1 of 3). Using a Multiple Occurrence Data Structure to Accumulate Totals – Example 1



```
C                                                
```

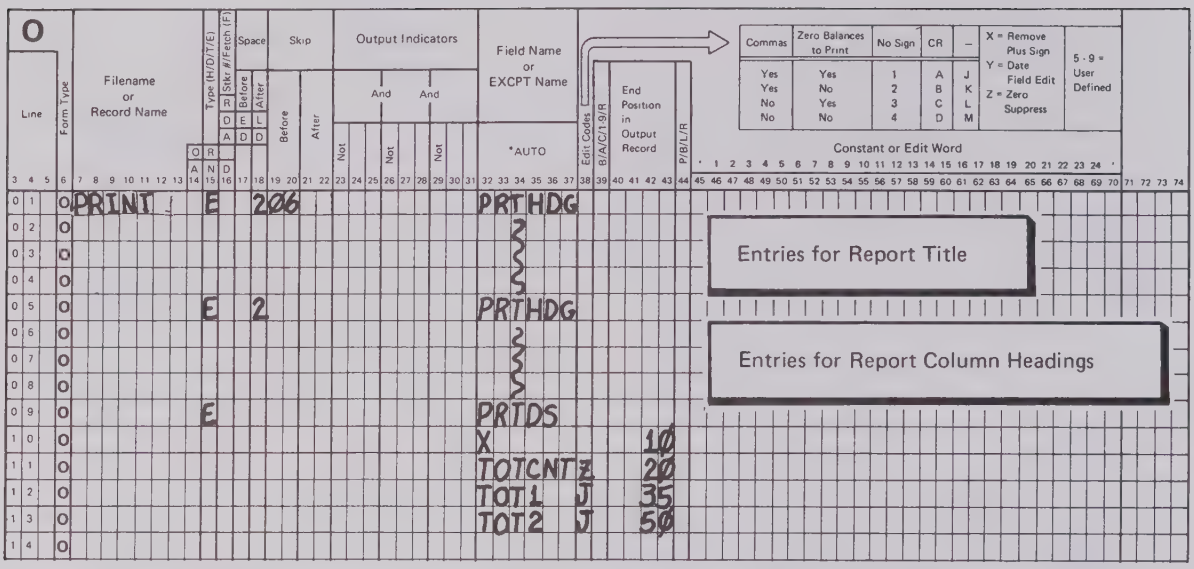
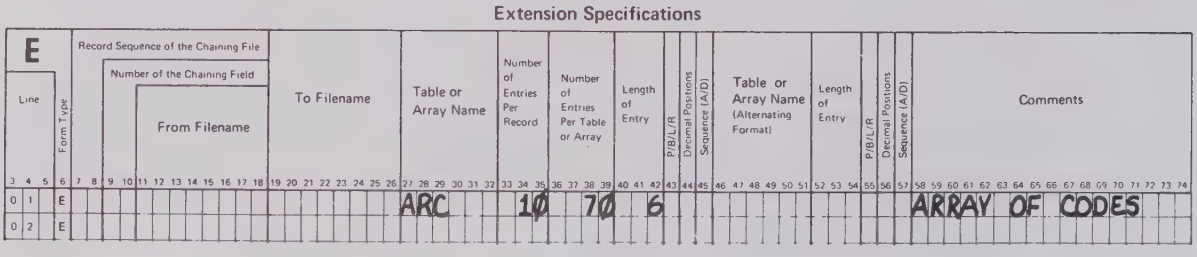


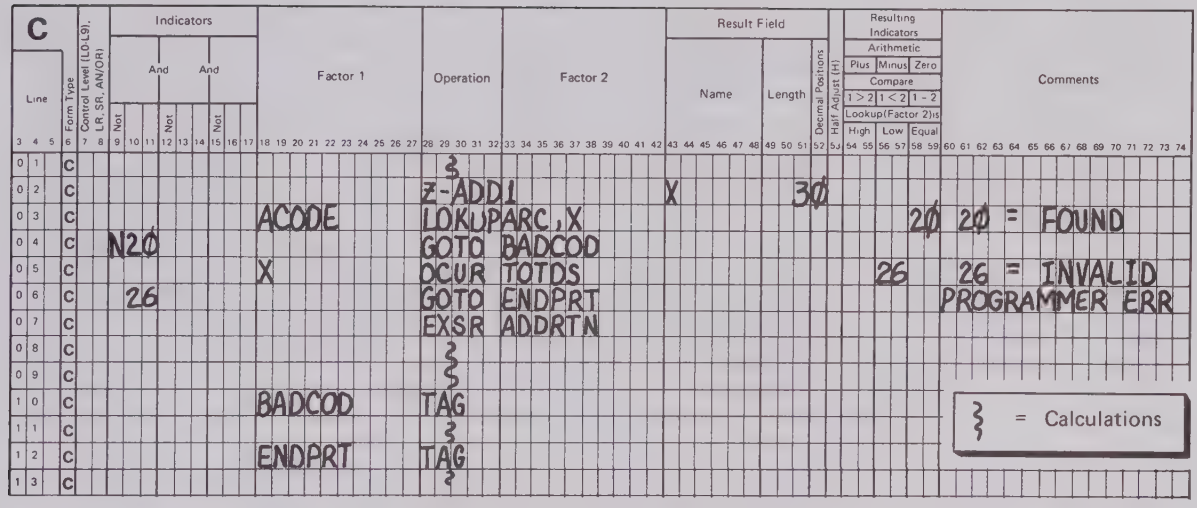
Figure 10-4 (Part 3 of 3). Using a Multiple Occurrence Data Structure to Accumulate Totals – Example 1



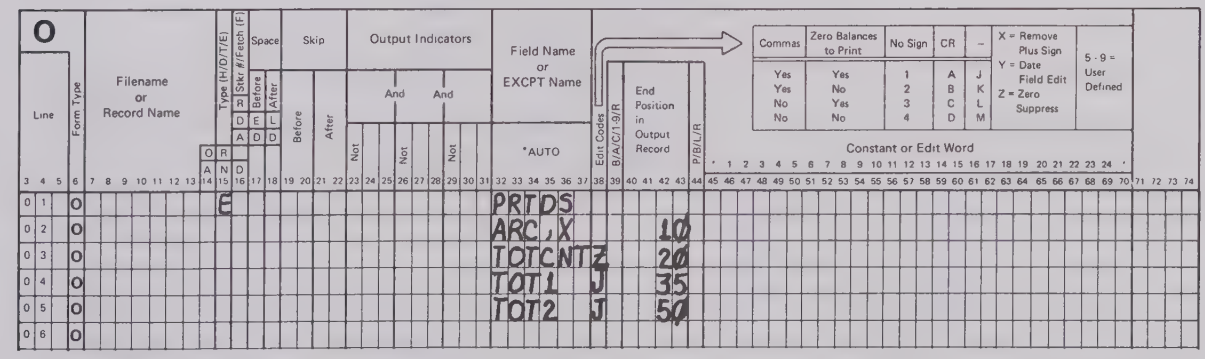
In this example, a multiple occurrence data structure, TOTDS, is again used to accumulate a series of totals for specific codes and the totals of each of the occurrences of the data structure are written. There are 70 codes.

A compile-time array, ARC, is specified in the extension specifications. It has 70 entries. There are 10 entries in each record and each array element is 6 positions in length. The relative location of the alpha code in the array (for example, the 37th entry) is used to set the current occurrence of the data structure.

Figure 10-5 (Part 1 of 2). Using a Multiple Occurrence Data Structure to Accumulate Totals – Example 2



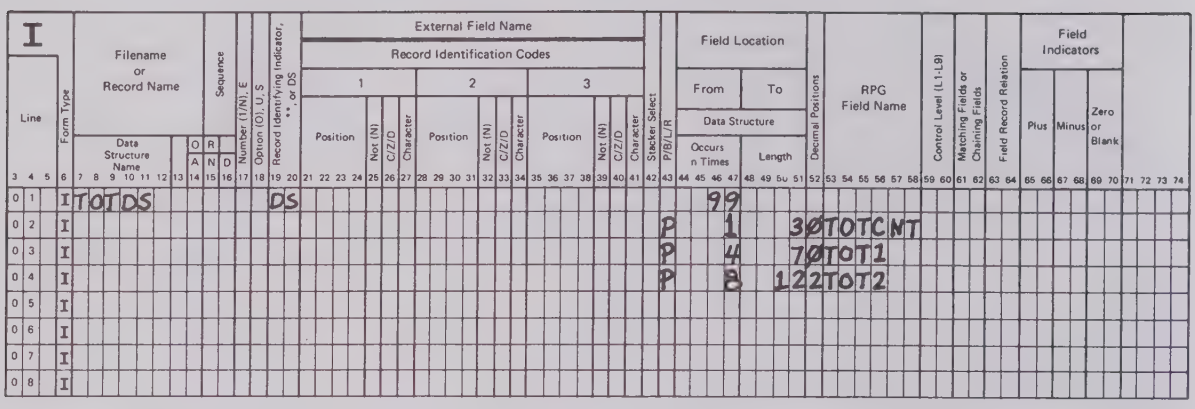
In the calculation specifications, the Z-ADD operation (line 02) sets field X to one. The LOKUP operation (line 03) starts at the first element of ARC and searches until it finds the first element equal to the code in ACODE. The ACODE field is a character field of 6 characters. The index value, X, is set to the position number of the element located. If LOKUP does not find an element equal to ACODE, indicator 20 is not set on and the GOTO operation conditioned by N20 branches to the BADCOD TAG (line 10). If LOKUP does find an element equal to ACODE, the OCUR operation (line 05) uses the value in X to set the current occurrence of TOTDS and the program branches to the ADDRTN subroutine (line 07), where a record count is made and input values are added to the data structure subfields. The ADDRTN subroutine is not shown. If the occurrence is outside the valid range for the data structure, indicator 26 is set on (line 05), and the program branches to the ENDPRT TAG (line 12).

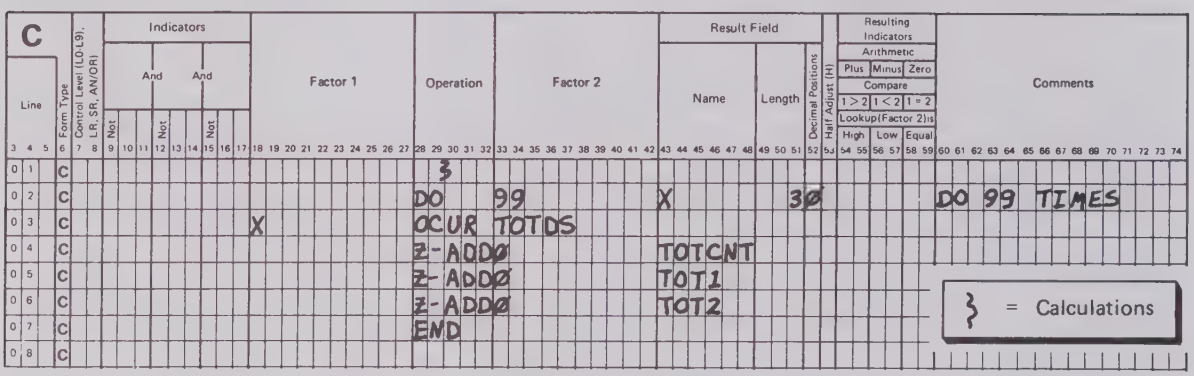


The calculations to print the data structure are not shown. Only part of the output specifications is shown. The PRTDS statement (line 01 of the output specifications) uses the value of field X, which contains the current occurrence of the data structure, as an index to print the corresponding alpha code.

Figure 10-5 (Part 2 of 2). Using a Multiple Occurrence Data Structure to Accumulate Totals – Example 2

Data Structure Examples





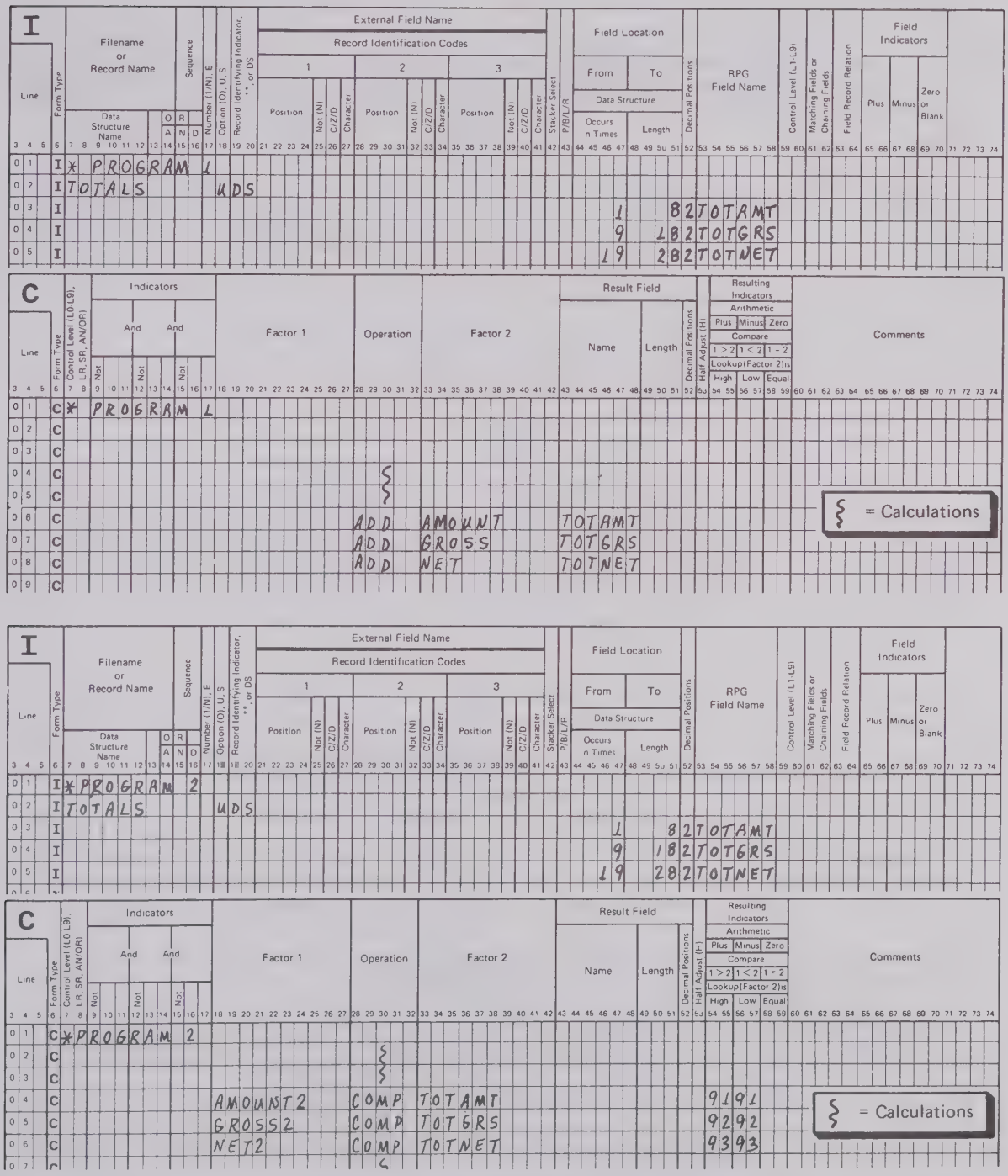
This example shows how to initialize data structure subfields that are in packed decimal format.

The system runs more efficiently if numeric fields are in packed rather than zoned decimal format because RPG/400 does all arithmetic operations in packed decimal format. When data structure subfields are in packed decimal format, they must be initialized to proper values *before* being used because RPG/400 initializes data structure subfields to blanks (hexadecimal 40's). If the data structure subfields are not initialized, the results of arithmetic operations are invalid.

The input specifications show that the TOTDS data structure has 99 occurrences as specified in positions 46 and 47. The subfields of the data structure (lines 02 through 04) are in packed decimal format as indicated by the P in position 43.

The do group (lines 02 through 07) is run 99 times. When the OCUR operation is run (line 03), the occurrence of TOTDS is set by the value of X and the program runs the initialization routine (lines 04 through 06). The Z-ADD operations initialize the data structure subfields to zeros.

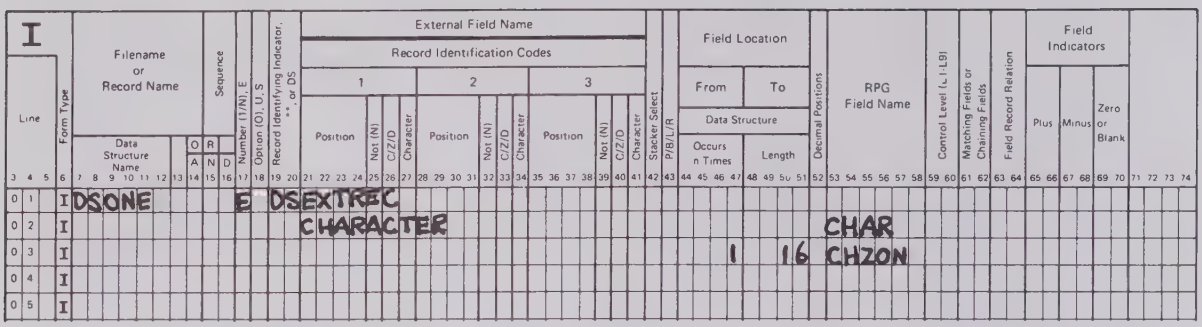
Figure 10-6. Initializing Data Structure Subfields in Packed Decimal Format



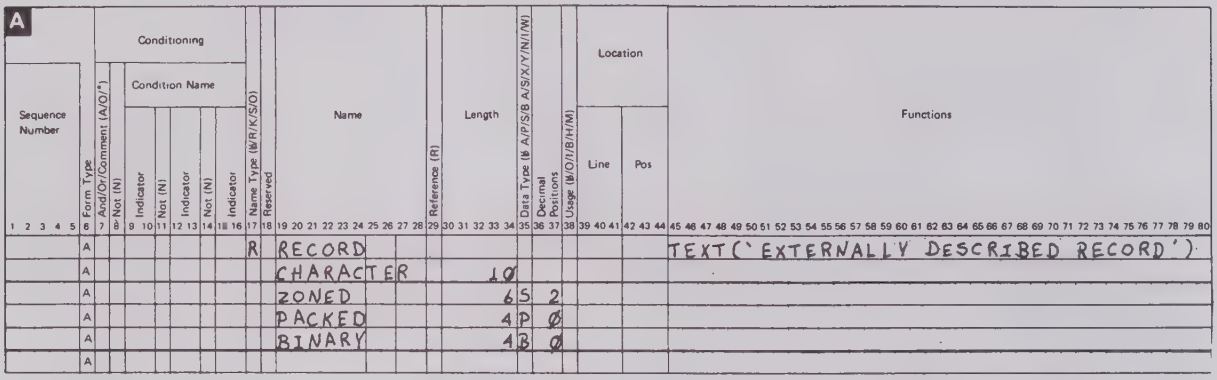
Both programs (1 and 2) use data area data structures (defined by the U in position 18 of the input specifications). Program 1 uses the subfields of the data structure to accumulate a series of totals. Program 2 then uses the totals in the subfields to do calculations.

Figure 10-7. Data Area Data Structures

Data Structure Examples



Line	Form Type	Filename or Record Name	Sequence	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, *, or DS	External Field Name / Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	Field Location: From	To	Data Structure: Occurs n Times	Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Minus	Zero or Blank
01	I	DSONE		E		DS	EXTREC																								
02	I						CHARACTER																		CHAR						
03	I																					1	16		CHZON						
04	I																														
05	I																														



Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (I/R/K/S/O)	Reserved	Name	Length	Reference (R)	Data Type (I/P/S/B/A/S/X/Y/N/1/N)	Decimal Positions	Usage (I/O/I1/B/H/M)	Location: Line	Pos	Functions
	A								R		RECORD								TEXT('EXTERNALLY DESCRIBED RECORD')
	A										CHARACTER	10							
	A										ZONED	6		S	2				
	A										PACKED	4		P	0				
	A										BINARY	4		B	0				
	A																		

On the data structure statement (line 01 of the input specifications), positions 7 through 12 contain the name of the data structure being defined (DSONE), position 17 contains an E to denote externally described, and positions 19 and 20 contain DS to denote data structure. Positions 21 through 30 contain the name of the file (EXTREC) whose first record format contains the field descriptions used as the subfield descriptions for this data structure (RECORD).

On the first data description specification, position 17 contains an R to denote record format and positions 19 through 28 contain the name of the record format (RECORD). On subsequent data description specifications, positions 19 through 28 contain the names of the fields (CHARACTER, ZONED, PACKED, and BINARY).

Fields in a data structure may also be redefined for program use. Fields CHARACTER and ZONED are also described as one field (CHZON) on line 03 of the input specifications.

In RPG/400, a field name can contain no more than 6 characters. Therefore, the field name CHARACTER is renamed CHAR on line 02 of the input specifications. The data structure then uses CHAR as the subfield name.

Figure 10-8. Renaming Subfields in an Externally Described Data Structure

Chapter 11. Communicating with Objects in the System

Calling Other Programs

RPG/400 provides for communication with other programs.

The CALL (call a program) operation code and the RETRN (return to caller) operation code allow an RPG/400 program to call other programs (for example, RPG/400 or CL) and to return to the caller. The PLIST (identify a parameter list) and PARM (identify parameters) operations allow the same data to be accessed by a calling and a called program.

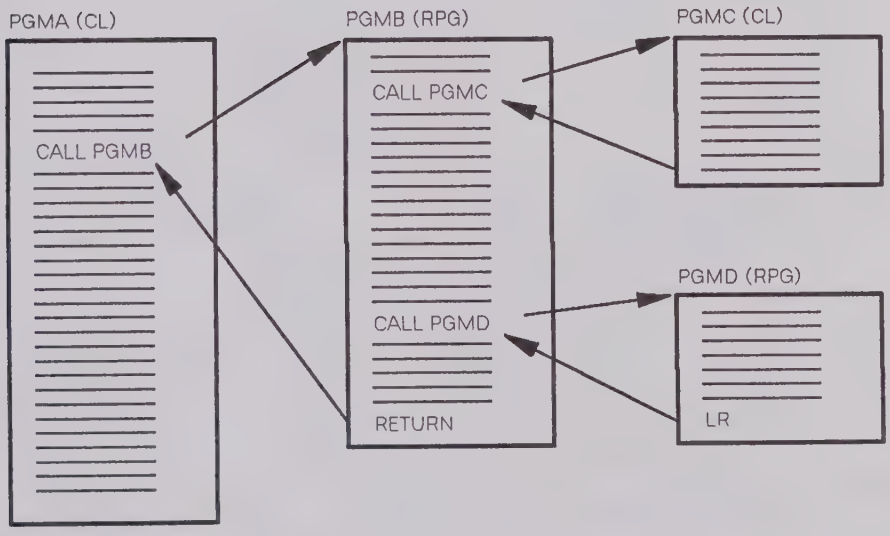


Figure 11-1. Calling RPG/400 and CL programs

Figure 11-1 shows a conceptual view of RPG/400 programs calling other programs (RPG/400 and CL) and CL programs calling other programs (RPG/400 and CL). See Figure 11-2 on page 11-2 for a coding example of an RPG/400 program calling another RPG/400 program using the CALL/RETRN function. See the *Programming: Control Language Programmer's Guide* for information about passing parameters between an RPG/400 program and a CL program.

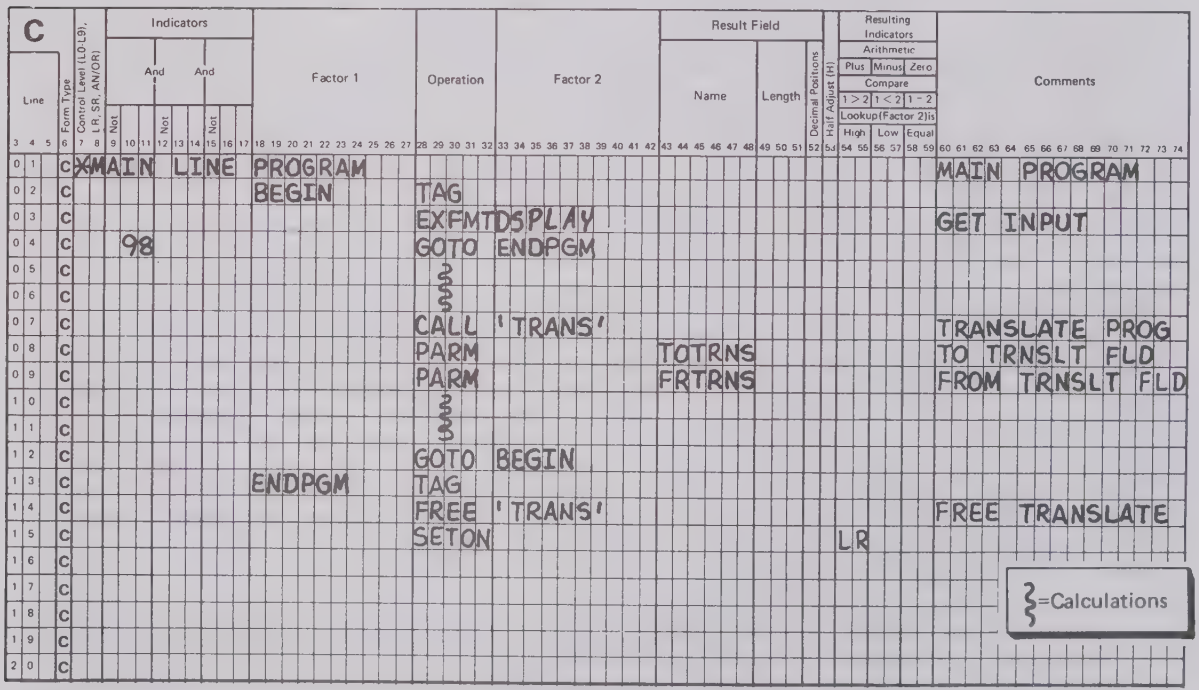
The CALL/RETRN function provides the following capabilities:

- PLIST and PARM(s) operation codes can be specified with the CALL operation to allow the same data to be accessed by a calling and a called program.

  When an RPG/400 program is called for the first time, the program is located, the fields are set up, and the program is given control. On each succeeding call, if the called program has not ended, all fields, indicators, and files in the called program are the same as they were when the program returned on the preceding

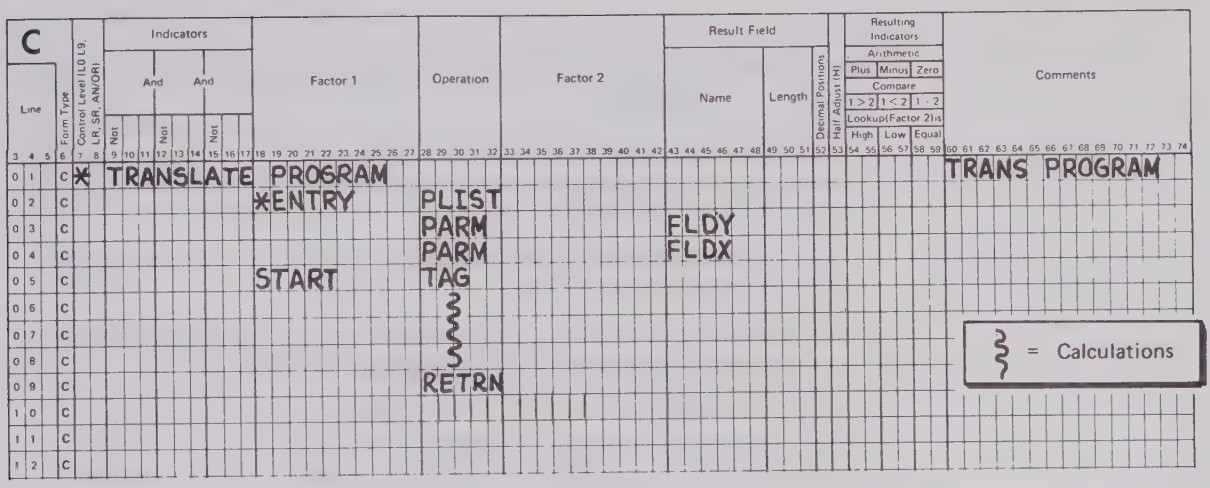
call. On each succeeding call, if the called program has ended or if FREE was specified, a fresh copy of the program is made available.

- The FREE operation code can be specified to remove a called program from the list of activated programs. If the program is called again, it functions as though it were being called for the first time. However, any files that are opened or any data areas that are locked by the called program are not affected by the FREE operation; the files or data areas are still allocated to the called program.
- The CALL operation can be dynamic; that is, the name of the program to be called can be supplied at run time.
- An explicit return is provided through the RETRN operation code.
- An implicit return is provided if the LR, RT, or H1 through H9 indicators are set on, or if the RPG/400 exception/error handling routine receives control when exception/errors occur.



This example shows an RPG/400 program (MAIN) using the CALL/RETRN function to call another RPG/400 program (TRANS). The EXFMT operation in the MAIN program (line 03) writes the DSPLAY record to the display screen. TOTRNS and FRTRNS are fields in the record. The work station user can key data into the TOTRNS field. The information in the TOTRNS field is to be translated by the TRANS program.

Figure 11-2 (Part 1 of 2). CALL/RETRN Function



When the CALL 'TRANS' operation in the MAIN program is processed (line 07), the FLDY and FLDX names in the TRANS program are used to access the data in the TOTRNS and FRTRNS fields in the parameter list specified in the MAIN program. Using this data, the TRANS program translates the TOTRNS field, which is called FLDY in the TRANS program, and places the result of the operation in the FLDX field. The RETRN operation in the TRANS program is then processed. (The translated field is called FRTRNS in the MAIN program.) A RETRN operation without the LR indicator on is specified in order to keep the program and all its work areas intact.

Return from the TRANS program is to the statement immediately following the last PARM statement in the MAIN program (line 10). The MAIN program completes the transaction (lines 10 and 11). When the GOTO operation is processed (line 12), the program branches back to the beginning of calculations. This loop continues until the work station user presses a command attention key that sets on indicator 98 to end the program. (On the DDS for the record format DSPLAY, a command attention key is associated with indicator 98.)

When indicator 98 is on, the program branches to the ENDPGM TAG statement and the FREE operation (line 14) frees the TRANS program. The MAIN program ends when LR is set on (line 15).

Figure 11-2 (Part 2 of 2). CALL/RETRN Function

CALL (Call a Program)

The CALL operation transfers control from the calling to the called program. A PLIST name is optional in the result field. If specified, it names a list of data that can be communicated between the calling program and the called program. If the called program accesses data in the calling program, and if the result field is blank, the CALL operation must be immediately followed by PARM operations.

When the CALL operation is processed, the calling program passes control to the called program. After the called program is run, control returns to the first processable statement after the CALL operation in the calling program. If an error occurs during processing of the CALL operation (for example, the called program is not found), the RPG/400 exception/error handling routine receives control. See "Exception/Error Handling" on page 4-35 for detailed information on the RPG/400 exception/error handling routine.

Remember the following when specifying CALL:.

- A program can contain multiple CALLs to the same program with the same or different PLISTs specified.
- The first CALL to a program causes program initialization. On subsequent CALLs to the same program, program initialization is bypassed unless the FREE operation was specified or the program was ended on a previous CALL.
- The addressing of parameters is limited to data formats common to the calling and called programs.
- When a calling program calls a non-RPG/400 program, the indicators in positions 56 and 57 are set on when the called program ends in error.
- An RPG/400 program cannot call itself or a program higher in the invocation stack. For example, if program A calls program B, then program B cannot call program A or B. If program B returns, with or without LR set on, and if program A then calls program C, program C can call program B but not program A or C.
- If, in factor 2 of a CALL operation, you use a literal to specify a program name without a library name, RPG/400 locates the program in your library list the first time the CALL operation is processed. When that CALL operation calls the same program again, it does not search your library list. If you want your library list to be searched each time the CALL operation is processed, specify the program name in a field rather than with a literal.

PLIST (Identify a Parameter List) and PARM (Identify Parameters)

The PLIST and PARM operations are nonprocessable calculation operations that can be used with CALL. The PLIST operation:

- Defines a name by which the following list of parameters (PARMs) can be specified in a CALL operation
- Defines the entry parameter list (*ENTRY PLIST) in a called program.

Factor 1 of the PLIST statement must contain the PLIST name. This name can be specified in the result field of one or more CALL operations. If the parameter list is the entry parameter list of a called program, factor 1 must contain *ENTRY. Only one *ENTRY PLIST can be specified in a program.

The parameters comprising the PLIST are defined by the immediately following PARM operations. The result field of a PARM statement identifies the data that the called program can address. Linkage between the calling and called program is by address; therefore, the parameters are name independent.

Rules for Specifying PLIST

Remember the following when specifying a PLIST statement:

- If PLIST is specified, it must immediately be followed by the PARM(s) that apply to it. If no PARM statements follow a PLIST statement, the PLIST statement is invalid.
- Multiple PLIST statements can appear in a program.
- Only one *ENTRY PLIST can occur in a program.

- A PLIST and its associated PARMs can appear anywhere in calculations.

Rules for Specifying PARM

Remember the following when specifying a PARM statement:

- One or more PARM statements must immediately follow a PLIST statement.
- One or more PARM statements can immediately follow a CALL operation.
- If there are more parameters in the calling program than in the called program, the called program will not run.
- If there are more parameters in the called program than in the calling program, an error occurs when an unresolved parameter is used.
- Fields specified as parameters in an *ENTRY PLIST can be used at first-page (1P) time.
- When a multiple occurrence data structure is specified in the result field of a PARM statement, all occurrences of the data structure are passed as a single field.
- The result field of a PARM statement cannot contain:
  - *IN
  - *INxx
  - *IN,xx
  - A literal
  - A data area name
  - A label
  - A data area data structure name
  - A table name.

  In addition, an array element, a data structure subfield name, the name of a compile-time array, and the name of a program status or file information data structure (INFDS) or a data structure specified in a *NAMVAR DEFN are not allowed in the result field of PARMs specified for an *ENTRY PLIST. A field name can be specified only once in an *ENTRY PLIST.
- When parameters are passed to an RPG/400 program that is called through CL, the parameters can be specified on the command that calls the program.
- Factor 1 of a PARM statement cannot contain a literal.
- Factor 1 and factor 2 must be blank if the result field contains the name of a multiple occurrence data structure.

OS/400 Graphics Support

RPG/400 allows you to use the CALL operation to call OS/400 Graphics, which includes the Graphical Data Display Manager (GDDM, a set of graphics primitives for drawing pictures), and Presentation Graphics Routines (a set of business charting routines). Factor 2 must contain the literal 'GDDM'. (not a field name or array element). Use the PLIST and PARM operations to pass the following parameters:

- The name of the graphics routine you want to run.
- The appropriate parameters for the specified graphics routine. These parameters must be of the data type required by the graphics routine.

RPG/400 does not implicitly start or end OS/400 graphics routines.

For more information on OS/400 Graphics, graphics routines and parameters, see the *Programming: Graphical Data Display Manager Programming Guide* and the *Programming: Graphical Data Display Manager Programming Reference.*

FREE (Deactivate a Program)

The FREE operation code:

- Removes a program from the list of activated programs
- Frees static storage if you no longer require the program
- Ensures program initialization (first cycle processing) when a program is called.

FREE does not close files or unlock data areas. You are responsible for closing files and unlocking data areas in your own program. In an interactive environment, you can close files and unlock data areas by using the CL command RETURN (from level 1 of the command entry display) or SIGNOFF (see the *Programming: Control Language Reference* for the use of RETURN and SIGNOFF).

When the FREE operation is specified, the program named in factor 2 is released from the list of activated programs. If the program is called by the CALL operation again, it functions as though it were being called for the first time (first-cycle processing). If the FREE operation is not successful, the RPG/400 exception/error handling routine receives control. See "Exception/Error Handling" on page 4-35 for detailed information on the RPG/400 exception/error handling routine.

Calling Special Subroutines

The three special subroutines that are available in RPG/400 are:

- Message-Retrieving Subroutine (SUBR23R3)
- Moving Double-byte Data and Deleting Control Characters (SUBR40R3)
- Moving Double-byte Data and Adding Control Characters (SUBR41R3)

Note: For detailed information on the use of CALL and PARM operation codes, see Languages: RPG/400 Reference.

Message-Retrieving Subroutine (SUBR23R3)

The message-retrieving subroutine (SUBR23R3) allows you to retrieve messages from a user message member QUSERMSG. If you want to use other message files, you may use the CL command OVRMSGF to override the message file. After the message has been retrieved, it can be modified and written to an output file.

Linkage to SUBR23R3 is by the CALL operation code, and input parameters are passed to SUBR23R3 by PARM operation codes. To use SUBR23R3, specify CALL in columns 28 to 32 and SUBR23R3 in columns 33 to 42. Five PARM operation codes must be specified after the CALL operation with the following result-field entries:

Result Field	Description
Message Identity (MSGID)	If LEVEL = 1 or 2: Name of a 4-digit numeric field that will be prefixed with 'USR' to form the message identity of the message to be retrieved. or If LEVEL = 3 or 4: Name of a 7-position character field that contains the message identifier to be retrieved.
Text area	Name of the alphanumeric field or data structure into which the message text is read. The maximum length of a level-1 message is 132 characters and of a level-2 message is 1435 characters. (Data structures must be used when the message is more than 256 characters.)
Level	Name of a one-digit numeric field that designates the user message member level. A value of 1 or 3 in this field indicates a message level of 1; a value of 2 or 4 indicates a message level of 2. (Data structures must be used when the message is more than 256 characters.) The value of 1 or 2 indicates the MSGID field is a 4-digit numeric field, and the value of 3 or 4 indicates the MSGID is a 7-digit alphanumeric field.
Return Code	Name of a one-digit numeric field that contains the return codes. The return codes and their meanings are as follows: *Return Code* — *Meaning* 0 — Message was successfully retrieved with no truncation. 1 — Message was successfully retrieved; but it was truncated because the length of the text area was less than the message length. 2 — Message was not found. 3 — Message level was invalid. (Not 1,2,3, or 4) 4 — An invalid MSGID value was diagnosed. (The value was not 0000 to 9999.) 5 — Message file was not found, or you didn't have the right authority, or message text length exceeds the level-1 maximum length. 6 — An invalid TXTL value was diagnosed.
Text Length (TXTL)	Name of a 4-digit numeric field that contains the length of the text area defined in the calling program.

The text area, which is specified by the second PARM operation, is blanked before each attempt to retrieve a message; therefore, a blank text area is returned to the user program, for some conditions, when the return code value is 2 or greater. A

total of 132 (1435 for level-2 messages) positions in the text area are blanked unless the text area is less than 132 (1435 for level-2 messages) characters in length.

Moving Double-byte Data and Deleting Control Characters (SUBR40R3)

SUBR40R3 is a move and edit routine that moves the contents of one field to another field. If the S/O and S/I control characters are found as the first and last characters in the field, SUBR40R3 deletes them.

SUBR40R3 is called as shown in Figure 11-3.

IBM International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

GX21-9093- UM/060* Printed in U.S.A

Program		Keying Instruction	Graphic									Card Electro Number	Page 1 2 of	Program Identification 75 76 77 78 79 80
Programmer	Date		Key											

C

Line	Form Type	Control Level (L0-L9), LR, SR, AN/OR	Indicators: And / And — Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic: Plus	Minus	Zero	Compare: 1>2	1<2	1=2	Lookup(Factor 2): High	Low	Equal	Comments
01	C				CALL	SUBR40R3														
02	C				PARM		EMPNO	10												SENDING FIELD
03	C				PARM		SOCSEC	8												RECEIVING FIELD
04	C				PARM		RETCDE	1												RETURN CODES
05	C				PARM		SNDLEN	30												SENDING LENGTH
06	C				PARM		RECLEN	30												RECEIVING LEN
07	C																			

Figure 11-3. Calling SUBR40R3

If you want the receiving field to contain all the data that was present in the sending field, you must specify a length for the receiving field that is two positions less than the length of the sending field. This allows two positions for each double-byte character (or one for each EBCDIC character) while deleting the S/O and S/I control characters (and the two positions they occupied). If you specify a receiving field longer than the sending field minus two positions, all the data from the sending field is moved and the receiving field is padded on the left with blanks (1-byte EBCDIC blanks). If the receiving field is shorter than the sending field minus two positions, the data being moved is truncated on the left.

Five PARM fields must be specified when SUBR40R3 is called. The first two specify the sending and receiving fields for the move. The third field is where the return codes are written to indicate the status of the move operation. The fourth and fifth fields must be loaded with the lengths of the sending and receiving fields. These are the lengths of the fields specified on the first two PARMs for the call to SUBR40R3 (in Figure 11-3, you would need to load the lengths of EMPNO and SOCSEC). The return code field must be defined as a one-position alphanumeric field; the length fields must be defined as three-position numeric fields with zero decimal positions.

SUBR40R3 produces return codes to indicate the status of the move operation. The following list contains these return codes and their meanings:

Return Code	Explanation
0	Data moved; no errors.
1	Data moved; padding occurred.
2	Data moved; truncation occurred.:
3	Data moved; S/O and S/I control characters were not found.
4	Data not moved. Either an odd field length was found, a length of zero was found, the length was greater than 256, or an invalid character was found in the field length. Length specified in fourth and fifth parameters is greater than the field length of the first and second parameters respectively.

If more than one return code can be issued, only the highest return code is returned.

Moving Double-byte Data and Adding Control Characters (SUBR41R3)

SUBR41R3 is a move and edit routine that moves the contents of one field into another field. If the S/O and S/I control characters are not found in the first and last positions of the field, SUBR41R3 adds them to the field when it is moved.

SUBR41R3 is called as shown in Figure 11-4.

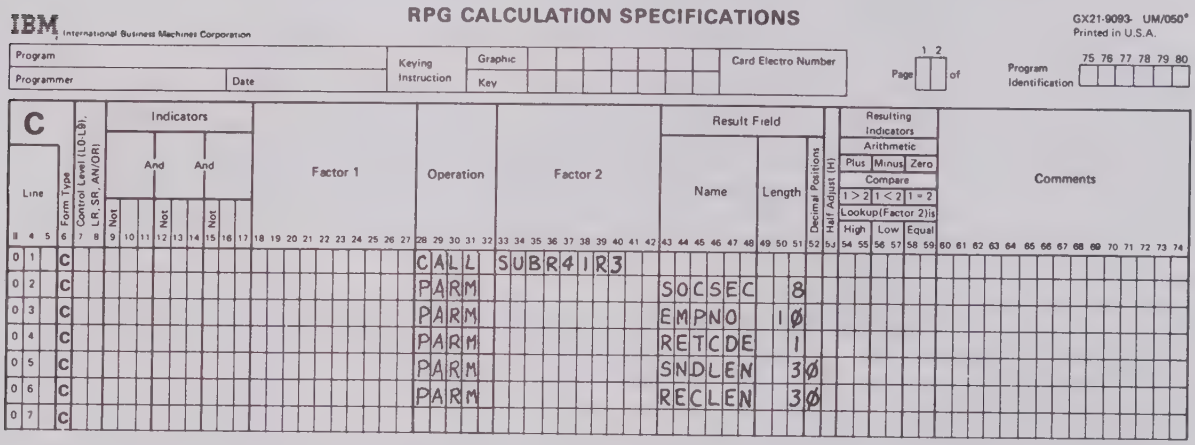


Figure 11-4. Calling SUBR41R3

If you want the receiving field to contain all the data that is in the sending field, you must specify the length of the receiving field to be two positions longer than the length of the sending field (to hold the S/O and S/I control characters). If you specify a receiving field that is longer than the sending field plus two, the data is padded on the left when it is moved into the receiving field. If the receiving field is shorter than the sending field plus two, the data is truncated on the left when it is moved. If the receiving field is specified either longer or shorter than the sending field plus two positions, the S/I control character is still placed in the correct position (the rightmost position).

Five PARM fields must be specified when SUBR41R3 is called. The first two specify the sending and receiving fields for the move. The third field is where the return codes are written to indicate the status of the move operation. The fourth and fifth fields must be loaded with the lengths of the sending and receiving fields. These are the lengths of the fields specified on the first two PARMs for the call to SUBR41R3 (in Figure 11-4, you would need to load the lengths of SOCSEC and EMPNO). The return code field must be defined as a one-position alphanumeric field; the length fields must be defined as three-position numeric fields with zero decimal positions.

SUBR41R3 produces return codes to indicate the status of the move. The following list contains these return codes and their meanings:

Return Code	Explanation
0	Data moved; no errors.
1	Data moved; padding occurred to left of S/I control character.
2	data moved; data truncated to left of S/I control character.
3	Data moved; S/O and S/I already present.
4	Data not moved. Either odd field length found, length of zero found, length greater than 256, or invalid character found in field length. Length is specified in fourth and fifth parameters is greater than the field length of the first and second parameters respectively.

If more than one return code can be issued, only the highest return code is issued.

Returning from a Called Program

An RPG/400 called program can return control to the caller in one of the following ways:

- With a normal end
- With an abnormal end
- Without an end.

A description of the ways to return from a called program follows.

For a detailed description of where the LR, H1 through H9, and RT indicators, and the RETRN operation are tested in the RPG/400 cycle, see "RPG/400 Program Cycle" on page 1-5.

A Normal End

A program ends normally and control returns to the caller when the LR indicator is on and an H1 through H9 indicator is not on. (For further information on the LR indicator, see Chapter 12, "Using Indicators.") The LR indicator can be set on by:

- The last record processed from a primary/secondary file during the RPG/400 cycle
- The programmer.

A program also ends normally if:

- The RETRN operation is processed, the H1 through H9 indicators are not on, and the LR indicator is on

- The RT indicator is on, the H1 through H9 indicators are not on, and the LR indicator is on.

See "RPG/400 Program Cycle" on page 1-5 for the sequence in which these operations and indicators are checked during the RPG/400 cycle.

When a program ends normally, the following occurs: parameters are moved from factor 2 to the result field; all arrays and tables with a "To file name" specified on the extension specifications, and all locked data area data structures are written out; any data areas locked by the program are unlocked; all files that are open are closed; a return code is set to indicate to the caller that the program has ended normally; and control returns to the caller. On the next call to the program, a fresh copy is available for processing.

An Abnormal End

A program ends abnormally and control returns to the caller when one of the following occurs:

- An H1 through H9 indicator is on, and the cancel option is taken when a message is issued.
- The cancel option is taken when an RPG/400 error message is issued.
- An *CANCL ENDSR statement in an *PSSR or INFSR subroutine is processed (for further information on the *CANCL return point for the *PSSR and INFSR subroutines, see "Exception/Error Handling" on page 4-35).
- An H1 through H9 indicator is on when a RETRN operation is processed.
- An H1 through H9 indicator is on when last record (LR) processing occurs in the RPG/400 cycle.

When a program ends abnormally, the following occurs: all files that are open are closed; any data areas locked by the program are unlocked; an error return code in the program status data structure is set to indicate to the caller that the program has ended abnormally; escape message RPG9001 is issued; and control returns to the caller. On the next call to the program, a fresh copy is available for processing. (For more information on the program status data structure, see "Exception/Error Handling" on page 4-35.)

Return without an End

A program can return control to the caller without ending when either the RETRN operation is processed or the RT indicator is set on, and the LR or H1 through H9 indicators are not on. The RETRN operation causes control to immediately return to the caller. The RT indicator causes control to return to the caller at a certain point in the cycle (after the H1 through H9 indicators and the LR indicator are tested). (For further information on the RT indicator, see Chapter 12, "Using Indicators.")

A program also returns without ending when something outside the program ends its activation. For example:

- RPG/400 program A calls another program (such as a CL program) that issues an escape message directly to the program calling A.

- A COBOL program calls an RPG/400 program that calls another COBOL program that ends using a STOP RUN. STOP RUN ends the COBOL run unit, which includes the RPG/400 program.

On the next call to a program that has not ended, all fields, indicators, and files in the program are the same as they were when the program returned on the preceding call.

You can use either the RETRN operation code or the RT indicator in conjunction with the LR indicator and the H1 through H9 indicators. When you do this, you must be aware of the testing sequence in the RPG/400 cycle for the RETRN operation, the RT indicator, the LR indicator, and the H1 through L9 indicators (see "RPG/400 Program Cycle" on page 1-5).

Data Areas

A data area is an object used to communicate data such as variable values between programs within a job and between jobs. A data area can be created and declared to a program before it is used in that program or job. For information on how to create and declare a data area, see the *Programming: Control Language Programmer's Guide*. RPG/400 does not support data areas defined by the CL command CRTDTAARA in which *LGL is specified as the TYPE parameter. The library that contains the data area must be specified in the library list.

RPG/400 provides access to a data area through a data area data structure, the data area operations IN and OUT, or a combination of the two. (For information on how to specify a data area data structure, see "Data Structures" on page 10-6. Figure 11-5 on page 11-15 shows a data area data structure and the IN and OUT operations (within the same program) accessing the same data area.

For a data area data structure, if the data area exists in a library that is specified in the library list, the data area is copied into the program. If the data area does not exist in a library that is specified in the library list, the name and length of the data structure are used to generate a data area in the job's temporary library (QTEMP).

RPG/400 retrieves and locks the contents of a data area at program initialization when a data area data structure is defined in the program. At the end of program, RPG/400 writes the data area data structure to the data area from which it came (temporary or permanent library) and unlocks the data area data structure. However, if a data area data structure is unlocked at the time RPG/400 does the update, RPG/400 does not write it at the end of program. At the end of job, the job's temporary library, QTEMP, is deleted.

The IN and OUT operations retrieve and write a data area. The lock capability is optional with these operations.

If the program calls another program that uses the same data area that the calling program uses, you must unlock the data area (with the UNLCK operation) before the other program is called. Two programs cannot simultaneously use the same data area for output.

A data area can be locked only once. An RPG/400 program cannot retrieve and lock a data area that has already been locked. Programs that attempt to retrieve and lock data areas include:

- Programs that use a data area as a data area data structure
- Programs that use an IN operation with *LOCK specified in factor 1.

The currently running program cannot lock the data area if:

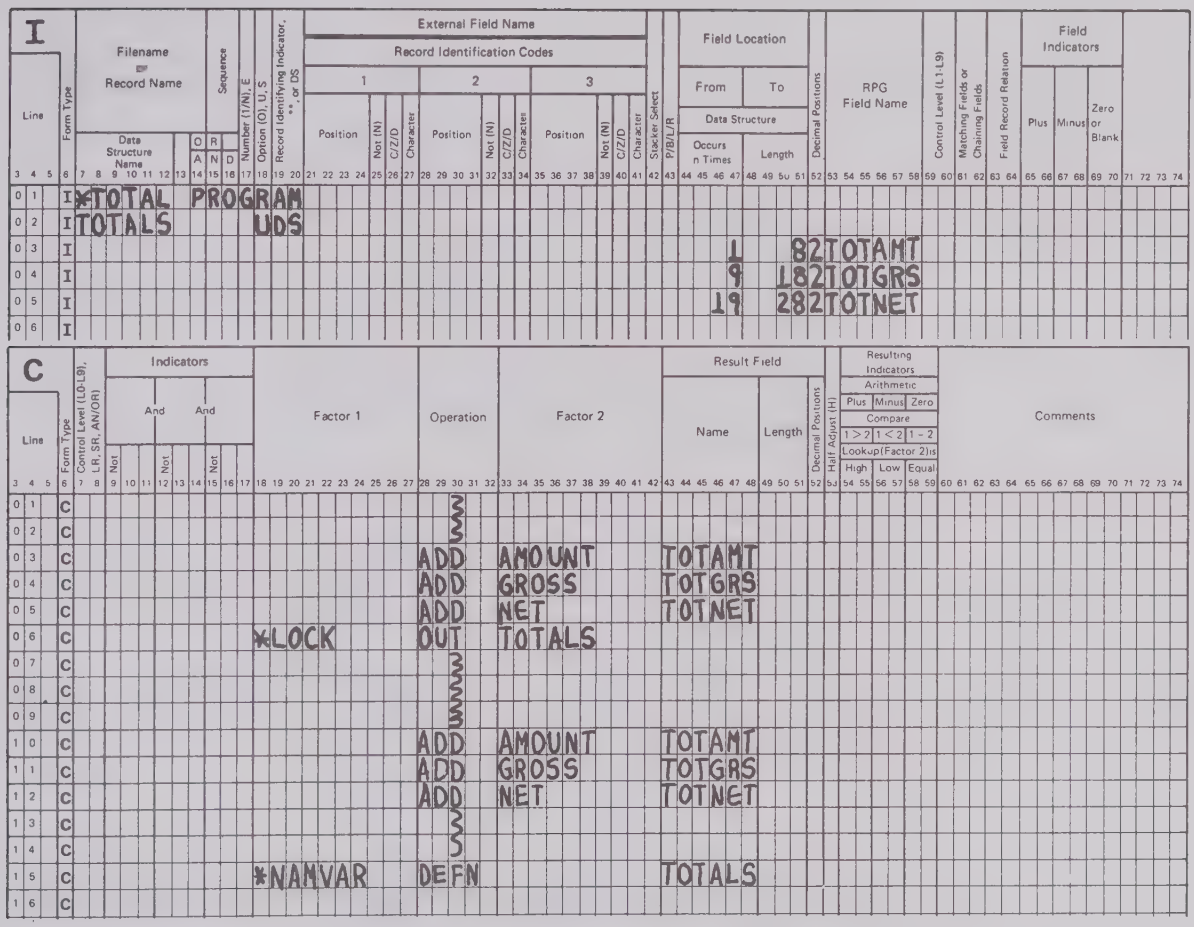
- The data area was used in a CL command ALCOBJ in the same or another routing step
- The data area was locked by a program that calls the current program.

To access a data area that has been locked with read allowed, retrieve it by using an IN operation with blanks in factor 1. In this case, the program can retrieve the data area but cannot change the data area by using the OUT operation.

RPG/400 uses the following lock states:

- An IN operation with *LOCK specified has an exclusive-allow-read (*EXCLRD) lock state.
- An IN operation without *LOCK specified has a shared-for-read (*SHRRD) lock state while transferring data. When the transfer is complete, RPG/400 releases the lock state.
- An OUT operation has an exclusive (*EXCL) lock state during the transfer of data and then that lock state is released. RPG/400 then releases the exclusive-allow-read (*EXCLRD) lock state established by the IN operation.

Another program's lock state on a data area may interfere with the operation of some or all of RPG/400's lock states. See the discussion of allocating resources in the *Programming: Control Language Programmer's Guide* for further information on which locks are compatible with other locks.



```
I
Line | Form Type | Filename / Record Name | Sequence | Number (1/N), E | Option (O), U, S | Record Identifying Indicator, **, or DS | External Field Name / Record Identification Codes 1 2 3 | Stacker Select | Field Location From To / Data Structure Occurs n Times, Length | Decimal Positions | RPG Field Name | Control Level (L1-L9) | Matching Fields or Chaining Fields | Field Record Relation | Field Indicators Plus Minus Zero or Blank
Data Structure Name | Option A | Sequence R | Sequence D | Position | Not (N) | C/Z/D | Character | P/B/L/R
3 4 5 | 6 | 7 8 9 10 11 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 20 | 21 22 23 24 | 25 | 26 | 27 | 28 29 30 31 | 32 | 33 | 34 | 35 36 37 38 | 39 | 40 | 41 | 42 | 43 | 44 45 46 47 | 48 49 50 51 | 52 | 53 54 55 56 57 58 | 59 60 | 61 62 | 63 64 | 65 66 | 67 68 | 69 70 | 71 72 73 74

0 1 I *TOTAL  PROGRAM
0 2 I TOTALS  UDS
0 3 I                                                                  1  82TOTAMT
0 4 I                                                                  9  182TOTGRS
0 5 I                                                                 19  282TOTNET
0 6 I

C
Line | Form Type | Control Level (L0-L9), LR, SR, AN/OR | Indicators And Not And Not And Not | Factor 1 | Operation | Factor 2 | Result Field Name | Result Field Length | Result Field Decimal Positions | Result Field Half Adjust (H) | Resulting Indicators Arithmetic Plus Minus Zero | Resulting Indicators Compare 1>2 1<2 1-2 | Resulting Indicators Lookup(Factor 2)is High Low Equal | Comments
3 4 5 | 6 | 7 8 | 9 10 11 12 13 14 15 16 | 17 | 18 19 20 21 22 23 24 25 26 27 | 28 29 30 31 32 | 33 34 35 36 37 38 39 40 41 42 | 43 44 45 46 47 48 | 49 50 51 | 52 | 53 | 54 55 56 57 58 59 | 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74

0 1 C
0 2 C
0 3 C                                                    ADD  AMOUNT  TOTAMT
0 4 C                                                    ADD  GROSS   TOTGRS
0 5 C                                                    ADD  NET     TOTNET
0 6 C                                  *LOCK             OUT  TOTALS
0 7 C
0 8 C
0 9 C
1 0 C                                                    ADD  AMOUNT  TOTAMT
1 1 C                                                    ADD  GROSS   TOTGRS
1 2 C                                                    ADD  NET     TOTNET
1 3 C
1 4 C
1 5 C                                  *NAMVAR           DEFN                 TOTALS
1 6 C
```

Figure 11-5. Data Area Data Structure and Data Area Operations

In this example, the data area data structure, TOTALS, is implicitly read and written by RPG/400 and explicitly updated.

At program initialization, RPG/400 implicitly retrieves and locks TOTALS. After calculations are processed on TOTALS (lines 3 through 5), the *LOCK OUT statement (line 6) updates and maintains the lock status of TOTALS. At end of program, RPG/400 implicitly writes and unlocks TOTALS.

Chapter 12. Using Indicators

The RPG/400 program cycle is built around indicators. To you, an indicator is a two-character entry on a specifications form that either is set on or off as the result of an operation or is used to condition (or control) when an operation is to be processed. To RPG/400, an indicator is an internal switch used by the program to remember when a certain event occurred and what to do when that event occurs.

Indicators are defined either by an entry on the specifications or by the RPG/400 program itself. The positions on the specifications form in which you define an indicator determines how the indicator is used. An indicator that has been defined can then be used to condition calculation and/or output operations.

The RPG/400 program sets and resets certain indicators at specific times during the program cycle. In addition, the status of most indicators can be changed by the SETON and SETOF operation codes. All indicators except indicators MR, 1P, KA through KN, and KP through KY can be set on with the SETON operation code; all indicators except MR and 1P can be set off with the SETOF operation code.

This chapter is divided into the following topics:

- Indicators defined on the RPG/400 specifications
- Indicators not defined on the RPG/400 specifications
- Use of indicators
- Indicators referred to as data.

Indicators Defined on RPG/400 Specifications

You can define an indicator on the RPG/400 specifications if it is specified as one of the following:

- Overflow indicator (positions 33 and 34 of the file description specifications)
- Record identifying indicator (positions 19 and 20 of the input specifications)
- Control level indicator (positions 59 and 60 of the input specifications)
- Field indicator (positions 65 through 70 of the input specifications)
- Resulting indicators (positions 54 through 59 of the calculation specifications)
- *IN array, *IN,xx array element or *INxx field (See "Indicators Referred to as Data" on page 12-32 for a description of how an indicator is defined when used with one of these reserved words.).

The defined indicator can then be used to condition operations in the program.

Overflow Indicators

An overflow indicator is defined by an entry in positions 33 and 34 of the file description specifications and is set on when the last line on a page has been printed or passed. Valid indicators are 0A through 0G, 0V, and 01 through 99. A defined overflow indicator can then be used to condition calculation and output operations. A detailed description of the overflow indicator and fetch overflow logic is given in "Printer Files" on page 6-17.

Record Identifying Indicators

A record identifying indicator is defined by an entry in positions 19 and 20 of the input specifications and is set on when the corresponding record type is selected for processing. That indicator can then be used to condition certain calculation and output operations.

The valid record identifying indicators are:

```
01-99
H1-H9
L1-L9
LR
U1-U8
RT
```

Record identifying indicators do not have to be assigned in any particular order. For an externally described file, a record identifying indicator is optional, but, if you specify it, it follows the same rules as for a program described file.

Generally, the indicators 01 through 99 as record identifying indicators. However, the control level indicators (L1 through L9) and the last record indicator (LR) can be used to cause certain total steps to be processed. If L1 through L9 are specified as record identifying indicators, lower level indicators are not set on.

When you select a record type for processing, RPG/400 sets on its corresponding record identifying indicator. All other record identifying indicators are off except when a file operation code is used to retrieve records from a file (see the following paragraph). The record identifying indicator is set on after the record is selected, but before the input fields are moved to the input area. The record identifying indicator for the new record is on during total time for the old record; therefore, calculations processed at total time using the fields of the old record cannot be conditioned by the record identifying indicator of the old record. You can set the indicators off at any time in the program cycle, or RPG/400 sets them off before the next primary or secondary record is selected.

If you use a file operation code on the calculation specifications to retrieve a record, the record identifying indicator is set on as soon as the record is retrieved from the file. The record identifying indicator is not set off until the point in the RPG/400 cycle where record identifying indicators are set off (see "RPG/400 Program Cycle" on page 1-5). Therefore, it is possible to have several record identifying indicators for the same file, as well as record-not-found indicators, set on concurrently if several operations are issued to the same file within the same RPG/400 program cycle.

Rules for Assigning Record Identifying Indicators

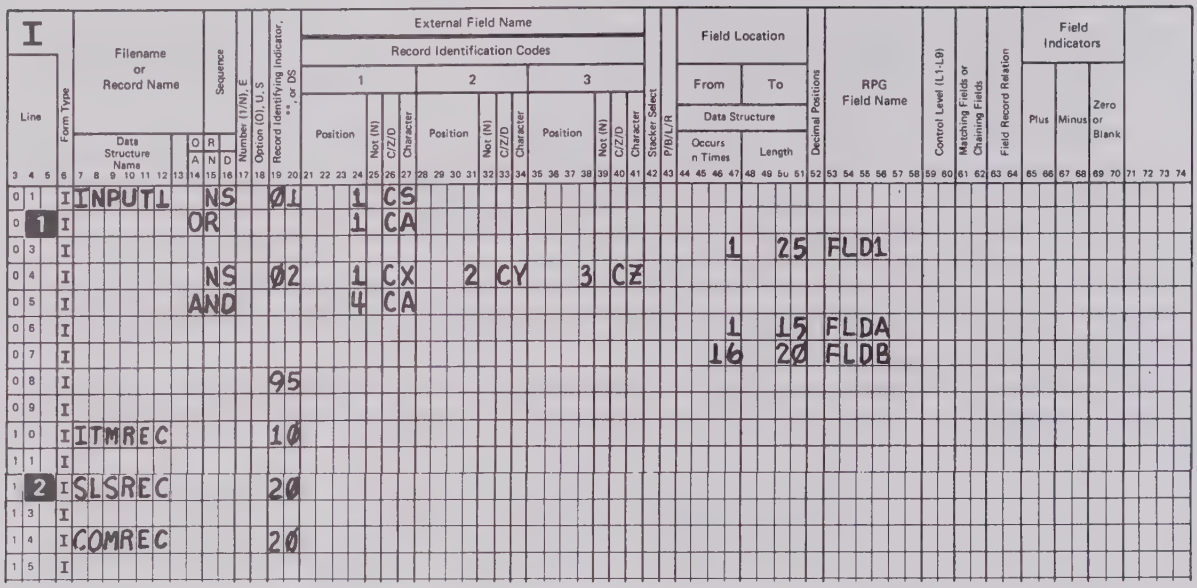
When you assign record identifying indicators to records in a program described file, remember the following:

- You can assign the same indicator to two or more different record types if the same operation is to be processed on all record types. To do this, you specify the record identifying indicator in positions 19 and 20, and specify the record identification codes for the various record types in an OR relationship.
- You can associate a record identifying indicator with an AND relationship, but it must appear on the first line of the group. Record identifying indicators cannot be specified on AND lines.
- An undefined record (a record in a program described file that was not described by a record identification code in positions 21 through 41) causes the program to halt.
- A record identifying indicator can be specified as a record identifying indicator for another record type, as a field indicator, or as a resulting indicator. No diagnostic message is issued, but this use of indicators may cause erroneous results.

When you assign record identifying indicators to records in an externally described file, remember the following:

- AND/OR relationships cannot be used with record format names; however, the same record identifying indicator can be assigned to more than one record.
- The record format name, rather than the file name, must be specified in positions 7 through 14.

For an example of record identifying indicators, see Figure 12-1.



Line	Form Type	Filename or Record Name	Sequence	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, ** or DS	External Field Name / Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	Field Location: From	Field Location: To	Data Structure: Occurs n Times	Data Structure: Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Field Indicators: Minus	Field Indicators: Zero or Blank
01	I	INPUT1	NS			01	1		CS																						
02	I		OR				1		CA																						
03	I																			1	25			FLD1							
04	I		NS			02	1		CX		2		CY		3		CZ														
05	I		AND				4		CA																						
06	I																			1	15			FLDA							
07	I																			16	20			FLDB							
08	I					95																									
09	I																														
10	I	ITMREC				10																									
11	I																														
12	I	SLSREC				20																									
13	I																														
14	I	COMREC				20																									
15	I																														

1 For this program described file, record identifying indicator 01 is set on if the record read contains an S in position 1 or an A in position 1. Record identifying indicator 02 is set on if the record read contains XYZA in positions 1 through 4. Record identifying indicator 95 is set on if any record is read that does not meet the requirements for record identifying indicator 01 or 02.

2 For this externally described file, record identifying indicator 10 is set on if the ITMREC record is read, and indicator 20 is set on if the SLSREC record or the COMREC record is read.

Figure 12-1. Examples of Record Identifying Indicators

Control Level Indicators (L1-L9)

A control level indicator is defined by an entry in positions 59 and 60 of the input specifications, designating an input field as a control field. It can then be used to condition calculation and output operations. The valid control level indicator entries are L1 through L9.

A control level indicator designates an input field as a control field. When a control field is read, the data in the control field is compared with the data in the same control field from the previous record. If the data differs, a control break occurs, and the control level indicator assigned to the control field is set on. You can then use control level indicators to condition operations that are to be processed only when all records with the same information in the control field have been read. Because the indicators stay on for both total time and the first detail time, they can also be used to condition total printing (last record of a control group) or detail printing (first record in a control group). Control level indicators are set off before the next record is read.

You can specify a control break after the first record containing a control field is read. The control fields in this record are compared to an area in storage that contains hexadecimal zeros. Because fields from two different records are not being compared, total calculations and total output operations are bypassed for this cycle.

Control level indicators are ranked in order of importance with L1 being the lowest and L9 the highest. All lower level indicators are set on when a higher level indicator is set on as the result of a control break. However, the lower level indicators can be used in the program only if they have been defined. For example, if L8 is set on by a control break, L1 through L7 are also set on. The LR (last record) indicator is set on when the input files are at end of file. LR is considered the highest level indicator and forces L1 through L9 to be set on.

You can also define control level indicators as record identifying or resulting indicators. When you use them in this manner, the status of the lower level indicators is not changed when a higher level indicator is set on. For example, if L3 is used as a resulting indicator, the status of L2 and L1 would not change if L3 is set on.

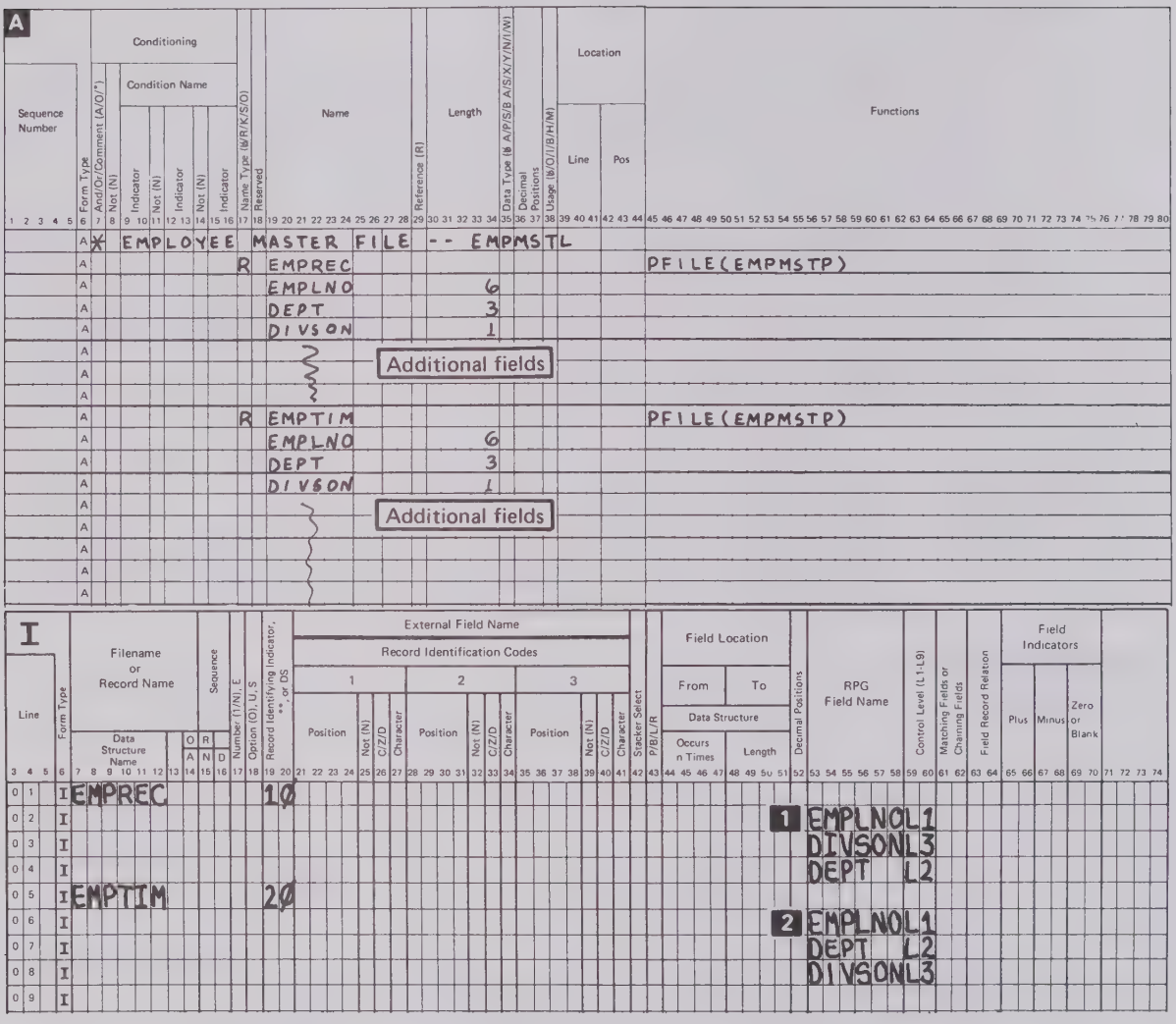
The importance of a control field in relation to other fields determines how you assign control level indicators. For example, data that demands a subtotal should have a lower control level indicator than data that needs a final total. A control field containing department numbers should have a higher control level indicator than a control field containing employee numbers if employees are to be grouped within departments (see Figure 12-2 on page 12-6).

Rules for Control Level Indicators

When you assign control level indicators, remember the following:

- You can specify control fields only for primary or secondary files.
- You cannot specify control fields for full procedural files or for binary, or look-ahead fields.
- You cannot use control level indicators when an array name is specified in positions 53 through 58 of the input specifications; however, you can use control level indicators with an array element.
- Control level compare operations are processed for records in the order in which they are found, regardless of the file from which they come.
- If you use the same control level indicator in different record types or in different files, the control fields associated with that control level indicator must be the same length (see Figure 12-2 on page 12-6).
- Record positions in control fields assigned different control level indicators can overlap in the same record type (see Figure 12-3 on page 12-7). However, you cannot assign more that 256 positions as control fields. For example, in Figure 12-3 on page 12-7, 15 positions have been assigned to control levels.
- Field names are ignored in control level operations. Therefore, fields from different record types that have been assigned the same control level indicator can have the same name.
- Control levels need not be written in any sequence. An L2 entry can appear before L1. All lower level indicators need not be assigned.
- If different record types in a file do not have the same number of control fields, unwanted control breaks can occur. Figure 12-4 on page 12-7 shows an example of how to avoid unwanted control breaks.

Indicators Defined on RPG/400 Specifications



1 In this example, control level indicators are defined for three fields in two different records in an externally described file. The data description specifications for the file are shown above. Three control level indicators are assigned. The names of the control fields (DIVSON, DEPT, EMPLNO) give an indication of their relative importance. The division (DIVSON) is the most important group. It is given the highest control level indicator used (L3). The department (DEPT) ranks below the division; L2 is assigned to it. The employee field has the lowest control level indicator (L1) assigned.

2 The same control level indicators can be used for different record types. However, the control fields having the same indicators must be the same length. For records in an externally described file, the field attributes are defined in the external description.

Figure 12-2. Control Level Indicators (Two Record Types)

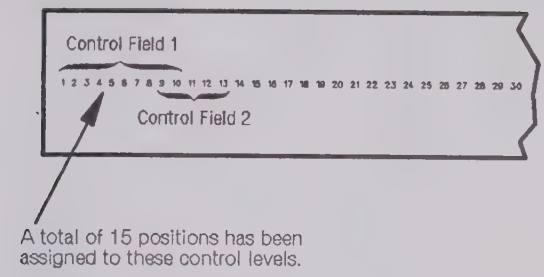
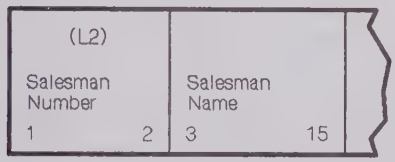
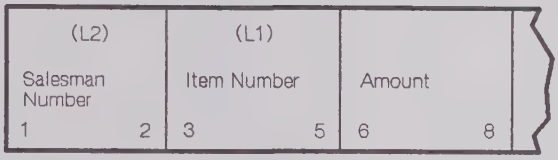


Figure 12-3. Overlapping Control Fields



Salesman Record

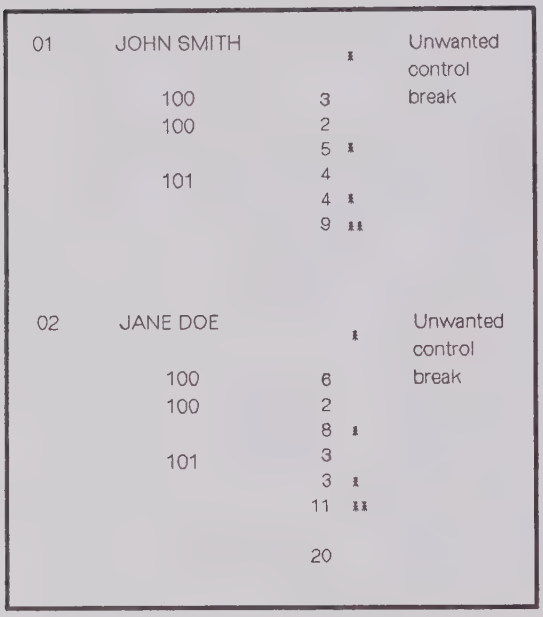


Item Record

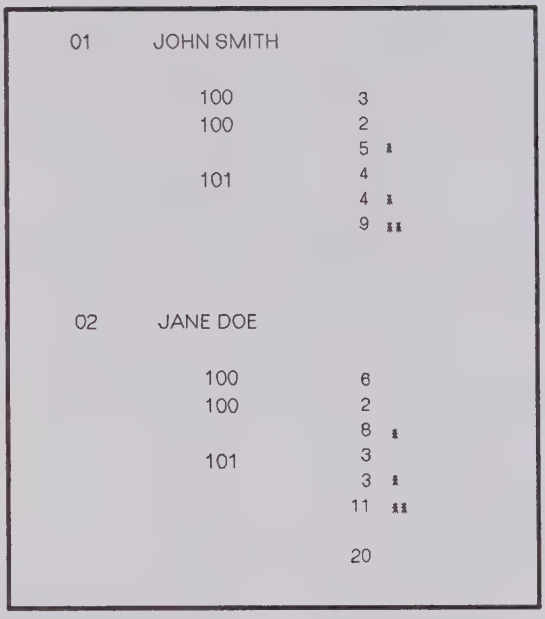
Different record types normally contain the same number of control fields. However, some applications require a different number of control fields in some records.

The salesman records contain only the L2 control field. The item records contain both L1 and L2 control fields. With normal RPG/400 coding, an unwanted control break is created by the first item record following the salesman record. This is recognized by an L1 control break immediately following the salesman record and results in an asterisk being printed on the line below the salesman record.

Figure 12-4 (Part 1 of 3). Unwanted Control Breaks

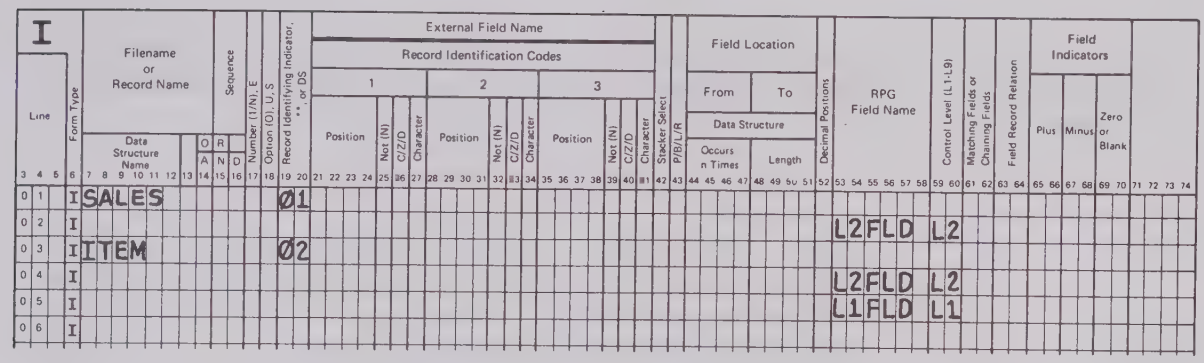


Output Showing Unwanted Control Level Break

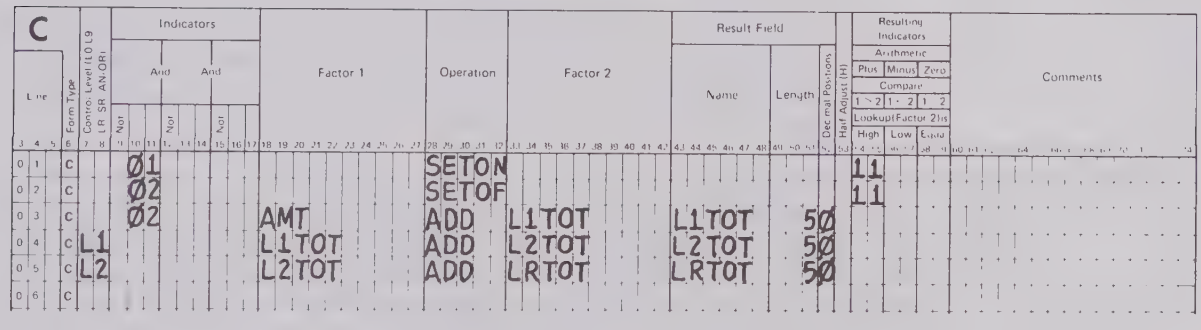


Corrected Output

Figure 12-4 (Part 2 of 3). Unwanted Control Breaks

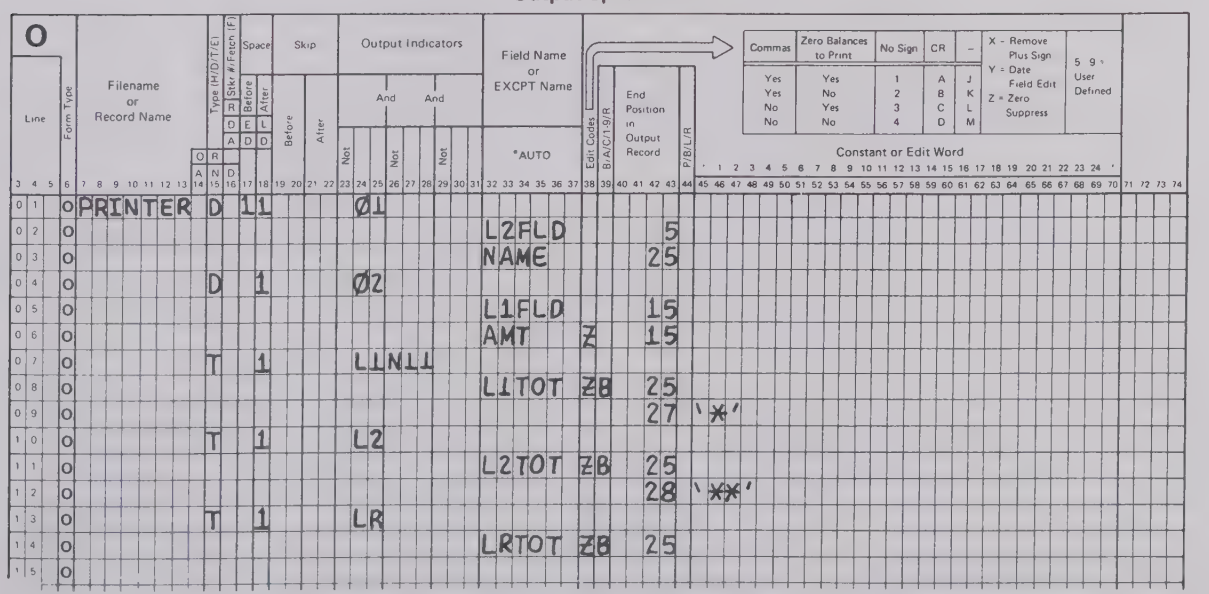


Calculation Specifications



This coding prevents the unwanted control break. Line 01 of the calculation specifications sets indicator 11 on when the salesman record is read. When the next item record causes an L1 control break, no total output is printed because indicator 11 is on (line 07 of output specifications). Detail calculations are then processed for the item record, and line 02 of the calculation specifications sets indicator 11 off. This allows the normal L1 control break to occur.

Output Specifications



- Numeric control fields are compared in zoned decimal format. Packed numeric input fields are converted to zoned decimal format according to the formula:

$$d = 2n-1$$

(d = number of digits in the field,
n = length of the input field).

The number of digits in a packed numeric field is always odd; therefore, when a packed numeric field is compared with a zoned decimal numeric field, the zoned field must have an odd length.

- When numeric control fields with decimal positions are compared to determined whether a control break has occurred, they are always treated as if they had no decimal positions. For instance, 3.46 is considered equal to 346.
- If you specify a field as numeric, only the digit portion determines whether a control break has occurred; that is, a field is always considered to be positive. For example, -5 is considered equal to +5.

Split Control Field

A split control field is formed when you assign more than one field in an input record the same control level indicator. For a program described file, the fields that have the same control level indicator are combined by the program in the order specified in the input specifications and treated as a single control field (see Figure 12-5). The first field defined is placed in the high-order (leftmost) position of the control field, and the last field defined is placed in the low-order (rightmost) position of the control field.

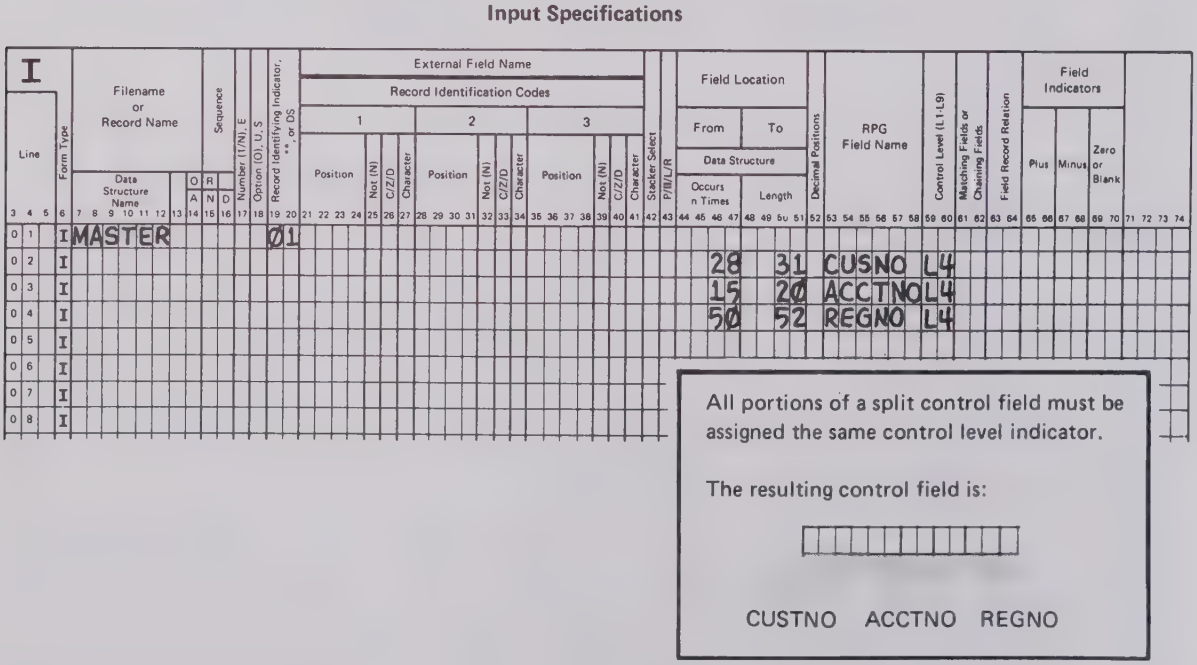


Figure 12-5. Split Control Fields

For an externally described file, fields that have the same control level indicator are combined in the order in which the fields are described in the data description specifications (DDS), not in the order in which the fields are specified on the input specifications. For example, if these fields are specified in DDS in the following order:

```
EMPNO

DPTNO

REGNO
```

and if these fields are specified with the same control level indicator in the following order on the input specifications:

```
REGNO L3

DPTNO L3

EMPNO L3
```

the fields are combined in the following order to form a split control field: `EMPNO DPTNO REGNO`.

Some special rules for split control fields are:

- For one control level indicator, you can split a field in some record types and not in others if the field names are different. However, the length of the field, whether split or not, must be the same in all record types.
- You can vary the length of the portions of a split control field for different record types if the field names are different. However, the total length of the portions must always be the same.
- A split control field can be made up of a combination of packed decimal fields and zoned decimal fields so long as the field lengths (in digits or characters) are the same.
- You must assign all portions of a split control field in one record type same field record relation indicator and it must be defined on consecutive specification lines.

Figure 12-6 on page 12-12 shows examples of the preceding rules.

Indicators Defined on RPG/400 Specifications

Record identification code = 1

Record identification code = 2

Record identification code = 3

The record identified by a 1 in position 95 has two split control fields:

FLD1A and FLD1B
FLD2A and FLD2B

The record with a 2 in position 95 has three split control fields:

FLD1A and FLD1B
FLD2A and FLD2B
FLD3A, FLD3B, and FLD3C

The third record type, identified by the 3 in position 95, also has three split control fields:

FLD1A and FLD1B
FLD2A and FLD2B
FLD3D and FLD3E

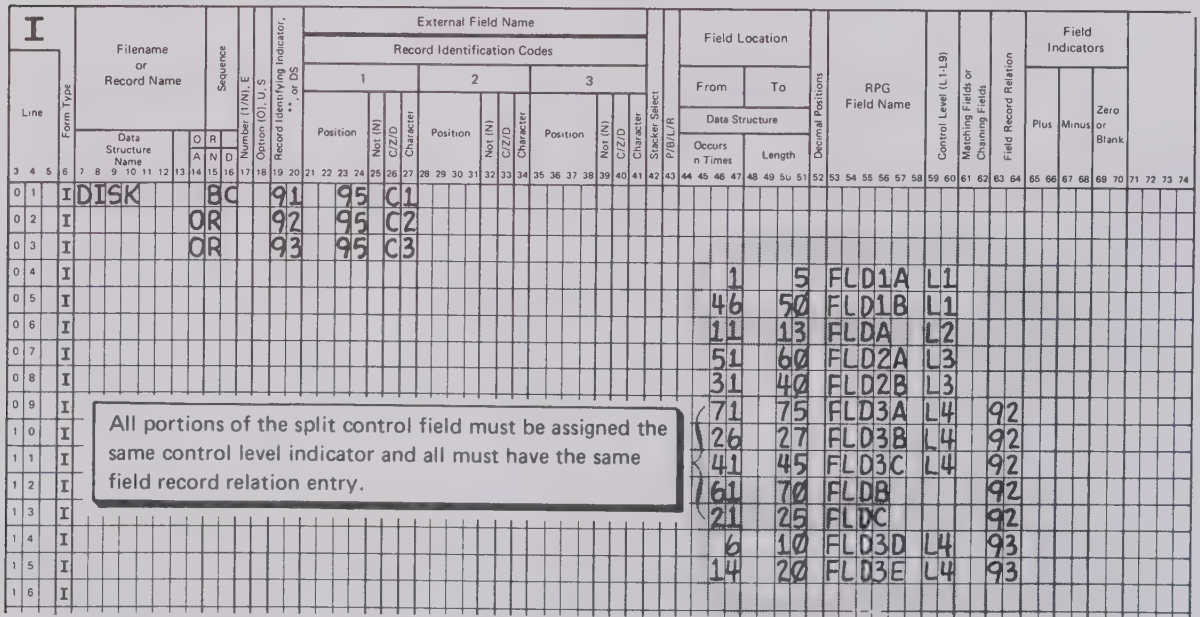


Figure 12-6. Split Control Fields – Special Rules

Field Indicators

A field indicator is defined by an entry in positions 65 and 66, 67 and 68, or 69 and 70 of the input specifications. The valid field indicators are:

```
01-99
H1-H9
U1-U8
RT
```

You can use a field indicator to determine if the specified field or array element is plus, minus, zero, or blank. Positions 65 through 68 are valid for numeric fields only; positions 69 and 70 are valid for numeric or character fields. An indicator specified in positions 65 and 66 is set on when the numeric input field is plus; an indicator specified in positions 67 and 68 is set on when the numeric input field is minus; and an indicator specified in positions 69 and 70 is set on when the numeric input field is zero or when a character input field is blank. You can then use the field indicator to condition calculation or output operations.

A field indicator is set on when the data for the field or array element is extracted from the record and the condition it represents is present in the input record. This field indicator remains on until another record of the same type is read and the condition it represents is not present in the input record, or until the indicator is set off as the result of a calculation.

You can use halt indicators (`H1` through `H9`) as field indicators to check for an error condition in the field or array element as it is read into the program.

Rules for Assigning Field Indicators

When you assign field indicators, remember the following:

- Indicators for plus, minus, zero, or blank are set off at the beginning of the program. They are not set on until the condition (plus, minus, zero, or blank) is satisfied by the field being tested on the record just read.
- Field indicators cannot be used with entire arrays or with look-ahead fields. However, an entry can be made for an array element.
- A numeric input field can be assigned two or three field indicators. However, only the indicator that signals the result of the test on that field is set on; the others remain off.
- If the same field indicator is assigned to fields in different record types, its status (on or off) is always based on the last record type selected.
- When different field indicators are assigned to fields in different record types, a field indicator remains on until another record of that type is read. Similarly, a field indicator assigned to more than one field within a single record type always reflects the status of the last field defined.
- The same field indicator can be specified as a field indicator on another input specification, as a resulting indicator, or as a record identifying indicator. No diagnostic message is issued, but this use of indicators could cause erroneous results.

- The same indicator can be specified in two positions, but the same indicator should not be used for all three positions. If the same indicator is specified in all three positions, the indicator is always set on when the record containing this field is selected.

Resulting Indicators

A resulting indicator is defined by an entry in positions 54 through 59 of the calculation specifications. The purpose of the resulting indicators depends on the operation code specified in positions 28 through 32. (See the individual operation code in *Languages: RPG/400 Reference* for a description of the purpose of the resulting indicators.) For example, resulting indicators can be used to test the result field after an arithmetic operation, to identify a record-not-found condition, to indicate an exception/error condition for a file operation, or to indicate an end-of-file condition.

The valid resulting indicators are:

```
01-99
H1-H9
OA-OG, OV
L1-L9
LR
U1-U8
KA-KN, KP-KY (valid only with SETOF)
RT
```

You can specify resulting indicators in three places (positions 54 through 59) of the calculation specifications. The positions in which the resulting indicator is defined determine the condition to be tested.

When a calculation is processed, the resulting indicators are set off, and, if the condition specified by a resulting indicator is satisfied, that indicator is set on. A resulting indicator can be used as a conditioning indicator on the same calculation line or in other calculations or output operations. When you use it on the same line, the prior setting of the indicator determines whether or not the calculation is processed. If it is processed, the result field is tested and the current setting of the indicator is determined (see Figure 12-7 on page 12-15).

Rules for Assigning Resulting Indicators

When assigning resulting indicators, remember the following:

- Resulting indicators cannot be used when the result field refers to an entire array.
- If the same indicator is used to test the result of more than one operation, the last operation processed determines the setting of the indicator.
- When L1 through L9 indicators are used as resulting indicators and are set on, lower level indicators are not set on. For example, if L8 is set on, L1 through L7 are not set on.

- If H1 through H9 indicators are set on when used as resulting indicators, the program halts unless the halt indicator is set off prior to being checked in the program cycle (see "RPG/400 Program Cycle" on page 1-5).
- The same indicator can be used to test for more than one condition depending on the operation specified.

Calculation Specifications

C	Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: And / And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Result Field: Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic: Plus / Minus / Zero	Resulting Indicators: Compare: 1>2 / 1<2 / 1=2	Resulting Indicators: Lookup(Factor 2)is: High / Low / Equal	Comments
	01	C													
	02	C			HRSWKD	SUB	40	OVERTM		30		1020			1
	03	C		N20	PAYRAT	MULT	40	PAY		62H					
	04	C		10	OVERTM	MULT	OVRRAT	OVRPAY		62H					
	05	C		10	OVRPAY	ADD	PAY	PAY							
	06	C		20	PAYRAT	MULT	HRSWKD	PAY							

Two resulting indicators **1** are used to test for the different conditions in a subtract operation. These indicators are used to condition the calculations that must be processed for a payroll job. Indicator 10 is set on if the hours worked (HRSWKD) are greater than 40 and is then used to condition all operations necessary to find overtime pay. Indicator 20 is set on if HRSWKD is less than 40. It is also used to condition other operations. In line 03 if indicator 20 is not on (the employee worked 40 or more hours), regular pay based on a 40-hour week is calculated. In line 06 if indicator 20 is on (employee worked less than 40 hours), pay based on less than a 40-hour week is calculated.

Figure 12-7. Resulting Indicators Used to Condition Operations

Indicators not Defined on the RPG III Specifications

Not all indicators that can be used as conditioning indicators in an RPG/400 program are defined on the specification forms. External indicators (U1 through U8) are defined by a CL command or by a previous RPG/400 program. Internal indicators (1P, LR, MR, and RT) are defined by the RPG/400 program cycle itself.

External Indicators

The external indicators are U1 through U8. These indicators can be set in a CL program or in an RPG/400 program. In a CL program, they can be set by the SWS (switch-setting) parameter on the CL commands CHGJOB (Change Job) or CRTJOBD (Create Job Description). In an RPG/400 program, they can be set as a resulting indicator or field indicator.

The status of the external indicators can be changed in the program by specifying them as resulting indicators on the calculation specifications or as field indicators on the input specifications. However, changing the status of the OS/400 job switches with a CL program during processing of an RPG/400 program has no effect on the copy of the external indicators used by the RPG/400 program. Setting the external indicators on or off in the program has no effect on file operations. File operations

function according to the status of the U1 through U8 indicators when the program is initialized. However, when a program ends normally with LR on, the external indicators are copied back into storage, and their status reflects their last status in the RPG/400 program. The current status of the external indicators can then be used by other programs.

Note: When using RETRN with the LR indicator off, you are specifying a return without an end and, as a result, no external indicators are updated. For more details about return without an end, see Chapter 11, "Communicating with Objects in the System."

Internal Indicators

Internal indicators are defined by the RPG/400 program itself. They include:

- First page indicator
- Last record indicator
- Matching record indicator

First Page Indicator (1P)

The first page (1P) indicator is set on by the RPG/400 program when the program starts running and is set off by the RPG/400 program after detail time output preceding the processing of the first record. The 1P indicator can be used to condition heading or detail records that are to be written at 1P time. Do not use the 1P indicator to condition output fields that require data from input records because the input data will not be available.

The 1P indicator cannot be used to condition total or exception output lines and should not be used in an AND relationship with control level indicators. The 1P indicator cannot be specified as a resulting indicator.

Last Record Indicator (LR)

In a program that contains a primary file, the last record indicator (LR) is set on by RPG/400 after the last record from a primary/secondary file has been processed, or it can be set on by the programmer.

The LR indicator can be used to condition calculation and output operations that are to be done at the end of the program. When RPG/400 sets the LR indicator on, all other control level indicators (L1 through L9) are also set on. If any of the indicators L1 through L9 have not been defined as control level indicators, as record identifying indicators, as resulting indicators, or by *INxx, the indicators are set on when LR is set on, but they cannot be used in other specifications.

In a program that does not contain a primary file, you can set the LR indicator on as one method to end the program. (For more information on how to end a program without a primary file, see "RPG/400 Program Cycle" on page 1-5.) To set the LR indicator on, you can specify the LR indicator as a record identifying indicator or a resulting indicator. If LR is set on during detail calculations, all other control level indicators are set on at the beginning of the next cycle. LR and the record identifying indicators are both on throughout the remainder of the detail cycle, but the record identifying indicators are set off before LR total time.

Matching Record Indicator (MR)

The matching record indicator (MR) is associated with the matching field entries M1 through M9. For a discussion of matching records, see "Primary/Secondary Multifile Processing" on page 6-33.

The MR indicator is set on when all the matching fields in a record of a secondary file match all the matching fields of a record in the primary file. It remains on during the complete processing of both primary and secondary records. It is set off when all total calculations, total output, and overflow for the records have been processed.

At detail time, MR always indicates the matching status of the record just selected for processing; at total time, it reflects the matching status of the previous record. If all primary file records match all secondary file records, the MR indicator is always on.

Use the MR indicator as a field record relation indicator, or as a conditioning indicator in the calculation specifications or output specifications to indicate operations that are to be processed only when records match. The MR indicator cannot be specified as a resulting indicator.

Using Indicators

Indicators that you have defined as overflow indicators, control level indicators, record identifying indicators, field indicators, resulting indicators, *IN, *IN,xx, *INxx, or those that are defined by RPG/400 can be used to condition files, calculation operations, or output operations. An indicator must be defined before it can be used as a conditioning indicator. The status (on or off) of an indicator is not affected when it is used as a conditioning indicator. The status can be changed only by defining the indicator to represent a certain condition.

File Conditioning

The file conditioning indicators are specified in positions 71 and 72 of the file description specifications. Only the external indicators U1 through U8 are valid for file conditioning. (An entry of UC can also be made in positions 71 and 72; however, UC is not an indicator.)

If the external indicator specified in positions 71 and 72 is off when the program is called, the file is not opened by RPG/400 and file operations for that file are ignored while the program is running. Primary and secondary input files are processed as if they were at end-of-file. The end-of-file indicator is set on for all READ operations to that file. Input, calculation, and output specifications for the file need not be conditioned by the external indicator.

Rules for File Conditioning

When you condition files, remember the following:

- A file conditioning entry can be made for input, output, update, or combined files.
- A file conditioning entry cannot be made for table or array input.

- Output files for tables can be conditioned by U1 through U8. If the indicator is off, the table is not written.
- A record address file can be conditioned by U1 through U8, but the file processed by the record address file cannot be conditioned by U1 through U8.
- If the indicator conditioning a primary file with matching records is off, the MR indicator is not set on by RPG/400.
- Output does not occur for an output, an update, or a combined file if the indicator conditioning the file is off.
- If the indicator conditioning an input, an update, or a combined file is off, the file is considered to be at end of file. The end-of-file indicator is set on for READ, READC, READE, and READP operations. CHAIN, EXFMT, SETGT, and SETLL operations are ignored and no indicators are set.

Field Record Relation Indicators

Field record relation indicators are specified in positions 63 and 64 of the input specifications. The valid field record relation indicators are:

```
01-99
H1-H9
MR
RT
L1-L9
U1-U8
```

Field record relation indicators cannot be specified for externally described files.

You use field record relation indicators to associate fields with a particular record type when that record type is one of several in an OR relationship. The field described on the specification line is available for input only if the indicator specified in the field record relation entry is on or if the entry is blank. If the entry is blank, the field is common to all record types defined by the OR relationship.

Assigning Field Record Relation Indicators

You can use a record identifying indicator (01 through 99) in positions 63 and 64 to relate a field to a particular record type. When several record types are specified in an OR relationship, all fields that do not have a field record relation indicator in positions 63 and 64 are associated with all record types in the OR relationship. To relate a field to just one record type, you enter the record identifying indicator assigned to that record type in positions 63 and 64 (see Figure 12-8 on page 12-20).

An indicator (01 through 99) that was previously defined in the program can also be used in positions 63 and 64 to condition movement of the field from the input area to the storage area.

Control fields, which you define with an L1 through L9 indicator in positions 59 and 60 of the input specifications, and match fields, which are specified by a match value (M1 through M9) in positions 61 and 62 of the input specifications, can also be related to a particular record type in an OR relationship if a field record relation indicator is

specified. Control fields or match fields in the OR relationship that do not have a field record relation indicator are used with all record types in the OR relationship.

If two control fields have the same control level indicator or two match fields have the same matching level value, a field record relation indicator can be assigned to just one of the match fields. In this case, only the field with the field record relation indicator is used when that indicator is on. If none of the field record relation indicators are on for that control field or match field, the field without a field record relation indicator is used. Control fields and match fields can only have entries of 01 through 99 or H1 through H9 in positions 63 and 64.

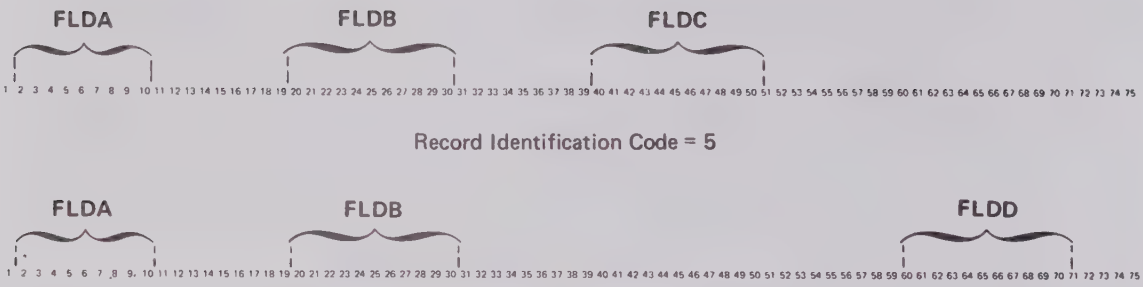
You can use positions 63 and 64 to specify that the program accepts and uses data from a particular field only when a certain condition occurs (for example, when records match, when a control break occurs, or when an external indicator is on). You can indicate the conditions under which the program accepts data from a field by specifying indicators L1 through L9, MR, or U1 through U8 in positions 63 and 64. Data from the field named in positions 53 through 58 is accepted only when the field record relation indicator is on.

External indicators are primarily used when file conditioning is specified in positions 71 and 72 of the file description specifications. However, they can be used even though file conditioning is not specified.

A halt indicator (H1 through H9) in positions 63 and 64 relates a field to a record that is in an OR relationship and also has a halt indicator specified in positions 19 and 20.

Remember the following points when you use field record relation indicators:

- Control level (positions 59 and 60) and matching fields (positions 61 and 62) with the same field record relation indicator must be grouped together.
- Fields used for control level (positions 59 and 60) and matching field entries (positions 61 and 62) without a field record relation indicator must appear before those used with a field record relation indicator.



The file contains two different types of records, one identified by a 5 in position 1 and the other by a 6 in position 1. The FLDC field is related by record identifying indicator 14 to the record type identified by a 5 in position 1. The FLDD field is related to the record type having a 6 in position 1 by record identifying indicator 16. This means that FLDC is found on only one type of record (that identified by a 5 in position 1) and FLDD is found only on the other type. FLDA is conditioned by indicator 07, which was previously defined elsewhere in the program. FLDB is found on both record types because it is not related to any one type by a record identifying indicator.

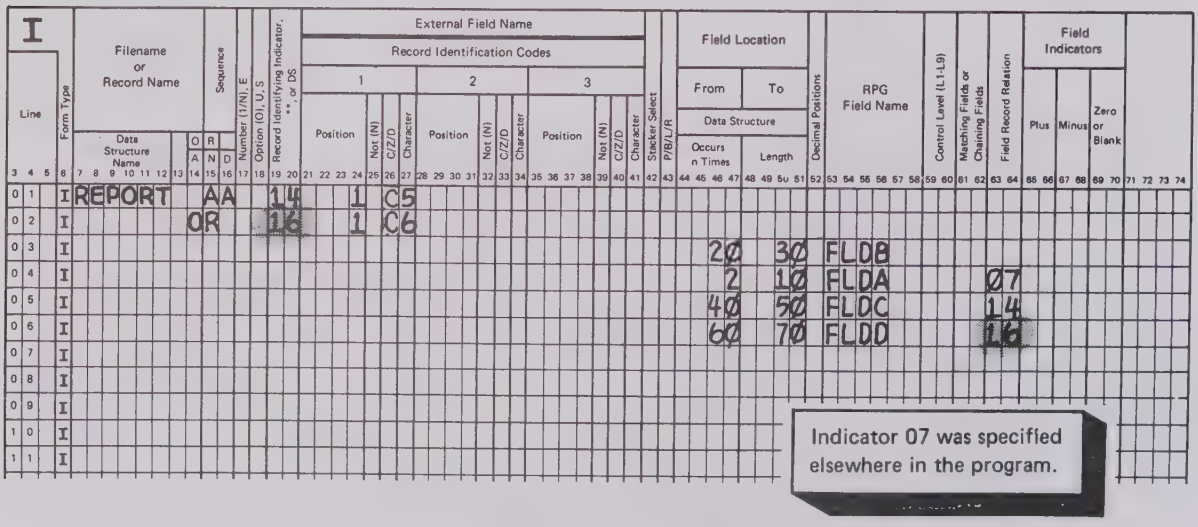


Figure 12-8. Field Record Relation

- Control level (positions 59 and 60) and matching fields (positions 61 and 62) with a field record relation indicator (positions 63 and 64) take precedence, when the indicator is on, over control level and matching fields of the same level without an indicator.
- Field record relations (positions 63 and 64) for matching and control level fields (positions 59 through 62) must be specified with record identifying indicators (01 through 99 or H1 through H9) from the main specification line or an OR relation line to which the matching field refers. If multiple record types are spec-

ified in an OR relationship, an indicator that specifies the field relation can be used to relate matching and control level fields to the pertinent record type.

- Noncontrol level (positions 59 and 60) and matching field (positions 61 and 62) specifications can be interspersed with groups of field record relation entries (positions 63 and 64).
- The MR indicator can be used as a field record relation indicator to reduce processing time when certain fields of an input record are required only when a matching condition exists.
- The number of control levels (L1 through L9) specified for different record types in the OR relationship can differ. There can be no control level for certain record types and a number of control levels for other record types.
- If all matching fields (positions 61 and 62) are specified with field record relation indicators (positions 63 and 64), each field record relation indicator must have a complete set of matching fields associated with it.
- If one matching field is specified without a field record relation indicator, a complete set of matching fields must be specified for the fields without a field record relation indicator.

Function Key Indicators

You can use function key indicators in a program that contains a WORKSTN device if the associated function keys are specified in data description specifications (DDS). Function keys are specified in DDS with the CFxx or CAxx keyword. For more information on using function key indicators with a WORKSTN file, see Chapter 9, "Using WORKSTN files."

The function key indicators correspond to function keys 1 through 24. Function key indicator KA corresponds to function key 1, KB to function key 2 . . . KY to function key 24 (see Table 12-2 on page 12-36).

Function key indicators that are set on can then be used to condition calculation or output operations. Function key indicators can be set off by the SETOF operation.

Return Indicator (RT)

You can use the return indicator (RT) to indicate to the internal RPG logic that control should be returned to the calling program. The test to determine if RT is on is made after the test for the status of LR and before the next record is read. If RT is on, control returns to the calling program. RPG/400 sets RT off when the program is called again.

Because the status of the RT indicator is checked after the halt indicators (H1 through H9) and LR indicator are tested, the status of the halt indicators or the LR indicator takes precedence over the status of the RT indicator. If both a halt indicator and the RT indicator are on, the halt indicator takes precedence. If both the LR indicator and RT indicator are on, the program ends normally.

RT can be set on as a record identifying indicator, a resulting indicator, or a field indicator. It can then be used as a conditioning indicator for calculation or output operations.

For a description of how RT can be used to return control to the calling program, see Chapter 11, "Communicating with Objects in the System."

Halt Indicators (H1-H9)

You can use the halt indicators (H1 through H9) to indicate errors that occur during the running of a program. The halt indicators can be set on as record identifying indicators, field indicators, or resulting indicators.

RPG tests the halt indicators at the *GETIN step of the RPG cycle (see "RPG/400 Program Cycle" on page 1-5). If a halt indicator is on, a message is issued to the requester. The following responses are valid:

- Set off the halt indicator and continue the program.
- Issue a dump and end the program.
- End the program with no dump.

If a halt indicator is on when a RETRN operation is processed or when the LR indicator is on, the called program ends abnormally. The calling program is informed that the called program ended with a halt indicator on. (For more information on abnormal ending, see Chapter 11, "Communicating with Objects in the System.")

For a detailed description of the steps that occur when a halt indicator is on, see the detailed flowchart of the RPG/400 cycle in "RPG/400 Program Cycle" on page 1-5.

Indicators Conditioning Calculations

Indicators that are used to specify the conditions under which a calculation is done are to be defined elsewhere in the program. Indicators to condition calculations can be specified in positions 7 and 8 and/or in positions 9 through 17.

Positions 7 and 8

You can specify control level indicators (L1 through L9 and LR) in positions 7 and 8 of the calculation specifications.

If positions 7 and 8 are blank, the calculation is processed at detail time, is a statement within a subroutine, or is a declarative statement. If indicators L1 through L9 are specified, the calculation is processed at total time only when the specified indicator is on. If the LR indicator is specified, the calculation is processed during the last total time.

Note: An L0 entry can be used to indicate that the calculation is a total calculation that is to be processed on every program cycle.

Positions 9-17

You can use positions 9 through 17 of the calculation specifications to specify indicators that control the conditions under which an operation is processed. The valid entries for positions 9 through 17 are:

```
01-99
H1-H9
MR
OA-OG, OV
L1-L9
LR
U1-U8
KA-KN, KP-KY
RT
```

Any indicator that you use in positions 9 through 17 must be previously defined as one of the following types of indicators:

- Overflow indicators (file description specifications, positions 33 and 34)
- Record identifying indicators (input specifications, positions 19 and 20)
- Control level indicators (input specifications, positions 59 and 60)
- Field indicators (input specifications, positions 65 through 70)
- Resulting indicators (calculation specifications, positions 54 through 59)
- External indicators
- Indicators set on by RPG/400, such as LR and MR
- *IN array, *IN,xx array element, or *INxx field (see "Indicators Referred to as Data" on page 12-32 for a description of how an indicator is defined when used with one of these reserved words).

You can specify one, two, or three indicators (positions 10 and 11, 13 and 14, and 16 and 17) on each line. If the indicator must be off to condition the operation, place an N before the appropriate indicator (positions 9, 12, and 15). When more than one indicator is specified in positions 9 through 17, the indicators are in an AND relationship. The indicators on one line or indicators in grouped AND/OR lines, plus the control level indicators (if specified in positions 7 and 8), must all be exactly as specified before the operation is done (see Figure 12-9 on page 12-24).

Lines of conditioning indicators in positions 9 through 17 can be placed in AND/OR relationships. AN/OR is specified in positions 7 and 8. A maximum of seven AND/OR lines can be specified in one group.

Figure 12-10 through Figure 12-12 show examples of conditioning indicators.

Some special considerations you should know when using conditioning indicators in positions 9 through 17 are as follows:

- Halt indicators can be used to end the program or to prevent the operation from being processed when a specified error condition is found in the input data

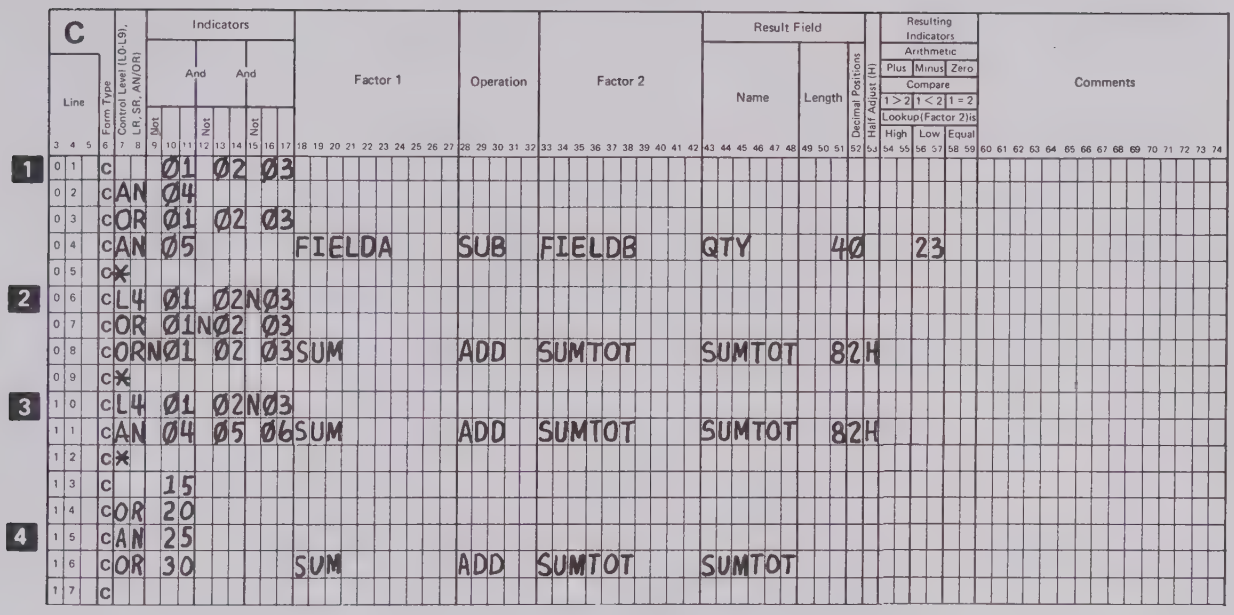
or in another calculation. Using a halt indicator is necessary because the record that causes the halt is completely processed before the program stops. Therefore, if the operation is processed on an error condition, the results are in error. A halt indicator can also be used to condition an operation that is to be done only when an error occurs.

- If LR is specified in positions 9 through 17, the calculation is done after the last record has been processed or after LR is set on.
- If a control level indicator is used in positions 9 through 17 and positions 7 and 8 are not used (detail time), the operation conditioned by the indicator is done only on the record that causes a control break or any higher level control break.
- If a control level indicator is specified in positions 7 and 8 (total time) and MR is specified in positions 9 through 17, MR indicates the matching condition of the previous record and not the one just read that caused the control break. After all operations conditioned by control level indicators in positions 7 and 8 are done, MR then indicates the matching condition of the record just read.
- If positions 7 and 8 and positions 9 through 17 are blank, the calculation specified on the line is done at detail calculation time.

Calculation Specifications

| C | | | | Indicators | | | |

Calculation Specifications



	Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: And	Indicators: And	Indicators: And	Factor 1	Operation	Factor 2	Result Field: Name	Result Field: Length	Result Field: Decimal Positions	Result Field: Half Adjust (H)	Resulting Indicators: Arithmetic: Plus	Resulting Indicators: Arithmetic: Minus	Resulting Indicators: Arithmetic: Zero	Resulting Indicators: Compare: 1>2	Resulting Indicators: Compare: 1<2	Resulting Indicators: Compare: 1=2	Resulting Indicators: Lookup(Factor 2)is: High	Resulting Indicators: Lookup(Factor 2)is: Low	Resulting Indicators: Lookup(Factor 2)is: Equal	Comments
1	01	C		01	02	03																	
	02	C	AN	04																			
	03	C	OR	01	02	03																	
	04	C	AN	05			FIELDA	SUB	FIELDB	QTY	40										23		
	05	C	*																				
2	06	C	L4	01	02	N03																	
	07	C	OR	01	N02	03																	
	08	C	ORN	01	02	03	SUM	ADD	SUMTOT	SUMTOT	82	H											
	09	C	*																				
3	10	C	L4	01	02	N03																	
	11	C	AN	04	05	06	SUM	ADD	SUMTOT	SUMTOT	82	H											
	12	C	*																				
	13	C		15																			
	14	C	OR	20																			
4	15	C	AN	25																			
	16	C	OR	30			SUM	ADD	SUMTOT	SUMTOT													
	17	C																					

1 AN and OR entries group lines of indicators. When indicators 01, 02, 03, and 04 are on, or when indicators 01, 02, 03, and 05 are on, the calculation is processed.

2 Three conditions cause the L4 total calculations to be processed: 01 and 02 are on, but not 03; or 01 and 03 are on, but not 02; or 02 and 03 are on, but not 01.

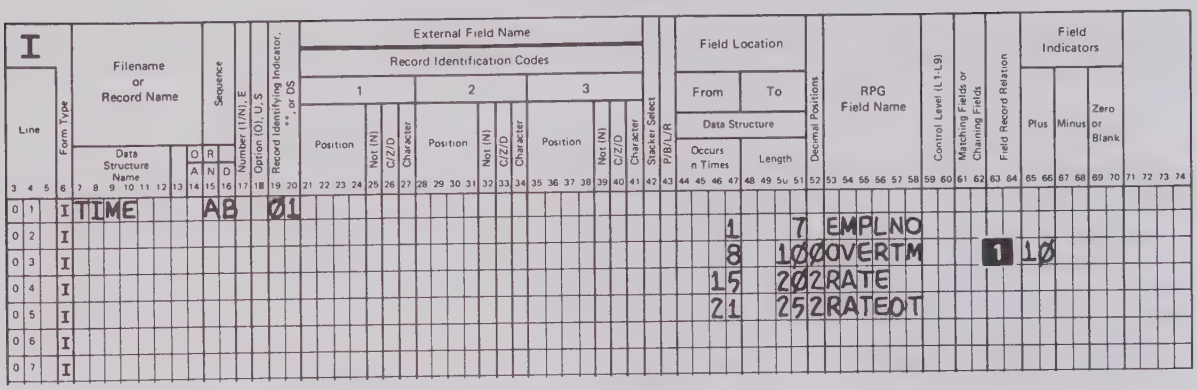
3 Seven conditioning indicators must be satisfied before the operation is processed: L4, 01, 02, 04, 05, and 06 must be on, and 03 must be off.

4 This calculation is done under one of three conditions: (1) when indicator 15 is on, (2) when indicators 20 and 25 are on, or (3) when indicator 30 is on. Each OR entry in positions 7 and 8 identifies the start of a new group of conditions. In this example, indicators 20 and 25 could have been coded in an AND relationship on the same line, as follows:

```
   15
OR 20 25
OR 30
```

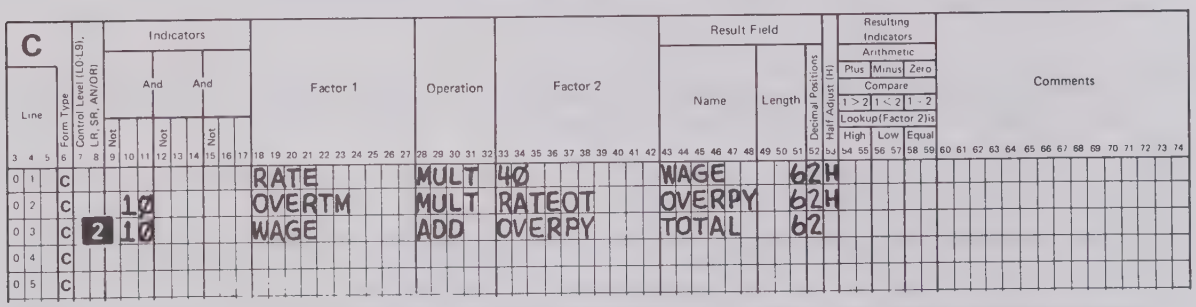
Figure 12-10. Calculation Conditioning Indicators in AND and OR Lines

Input Specifications



1 Field indicators can be used to condition operations. Assume the program is to find weekly earnings including overtime. The overtime field is checked to determine whether any overtime was put in. If the employee has worked overtime, the field is positive and indicator 10 is set on. In all cases the weekly regular wage is calculated. However, overtime pay is calculated only if indicator 10 is on (lines 02 and 03 of the calculation specifications).

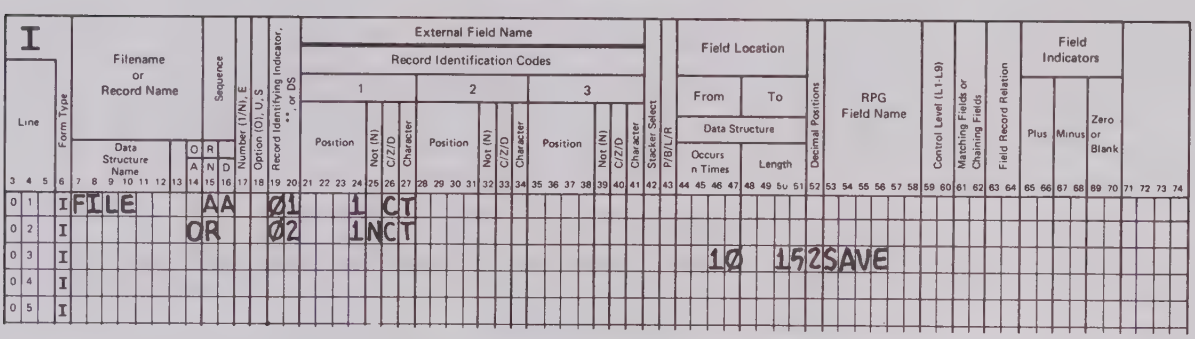
Input Specifications



2 Field indicator 10 was assigned on the input specifications. It is being used here to condition calculation operations.

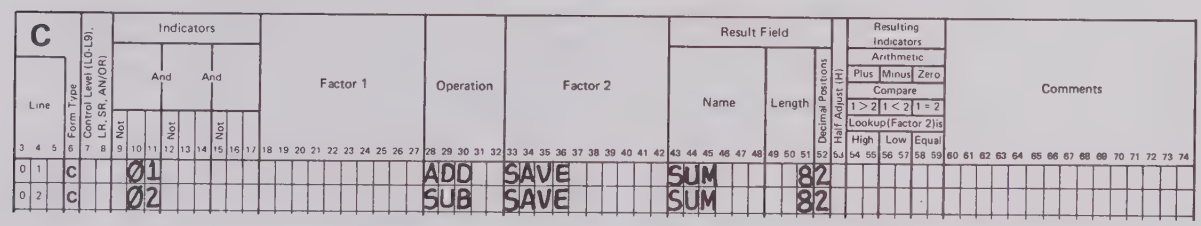
Figure 12-11. Conditioning Operations (Field Indicators)

Input Specifications



A record identifying indicator is used to condition an operation. When a record is read that has a T in position 1, the 01 indicator is set on. If this indicator is on, the field named SAVE is added to SUM. When a record having no T in position 1 is read, the 02 indicator is on. The subtract operation, conditioned by 02, is then done instead of the add operation.

Calculation Specifications



Record identifying indicators 01 and 02 are assigned on the input specifications. They are used here to condition calculation operations.

Figure 12-12. Conditioning Operations (Resulting Indicators)

Indicators Conditioning Output

Indicators that you use to specify the conditions under which an output record or an output field is written must be previously defined in the program. Indicators to condition output are specified in positions 23 through 31. The valid entries are:

```
01-99
H1-H9
1P
MR
OA-OG, OV
L1-L9
LR
U1-U8
KA-KN, KP-KY
RT
```

The indicators you use to condition output must be previously defined as one of the following types of indicators:

- Overflow indicators (file description specifications, positions 33 and 34)
- Record identifying indicators (input specifications, positions 19 and 20)

- Control level indicators (input specifications, positions 59 and 60)
- Field indicators (input specifications, positions 65 through 70)
- Resulting indicators (calculation specifications, positions 54 through 59)
- Indicators set by RPG/400, such as 1P and LR
- External indicators set prior to or during program processing
- *IN array, *IN,xx array element, or *INxx field (see "Indicators Referred to as Data" on page 12-32 for a description of how an indicator is defined when used with one of these reserved words).

If an indicator is to condition an entire record, you enter the indicator on the line that specifies the record type (see Figure 12-13 on page 12-29). If an indicator is to condition when a field is to be written, you enter the indicator on the same line as the field name (see Figure 12-13 on page 12-29).

Conditioning indicators are not required on output lines. If conditioning indicators are not specified, the line is output every time that type of record is checked for output. If you specify conditioning indicators, one indicator can be entered in each of the three separate output indicator fields (positions 24 and 25, 27 and 28, and 30 and 31). If these indicators are on, the output operation is done. An N in the position preceding each indicator (positions 23, 26, or 29) means that the output operation is done only if the indicator is not on (a negative indicator). No output line should be conditioned by all negative indicators; at least one of the indicators should be positive. If all negative indicators condition a heading or detail operation, the operation is done at the beginning of the program cycle when the first page (1P) lines are written.

You can specify output indicators in an AND/OR relationship by specifying AND/OR in positions 14 through 16. An unlimited number of AND/OR lines can be used. AND/OR lines can be used to condition output records, but they must not be used to condition fields. However, you can condition a field with more than three indicators by using the SETON operation in calculations. For example, if indicators 10, 12, 14, 16, and 18 are needed to condition an output field, in calculations use the SETON operation to set indicator 20 on if indicators 10, 12, and 14 are on. Then condition the output field with indicators 20, 16, and 18.

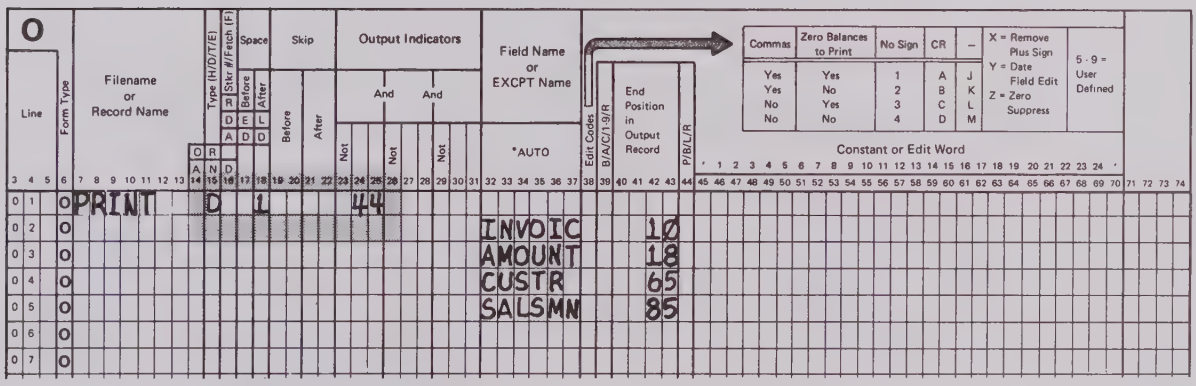
Other special considerations you should know about for output indicators are as follows:

- The first page indicator (1P) allows output on the first cycle before the primary file read, such as printing on the first page. The line conditioned by the 1P indicator must contain constant information used as headings or fields for reserved words such as PAGE and UDATE. The constant information is specified in the output specifications in positions 45 through 70. If 1P is used in an OR relationship with an overflow indicator, the information is printed on every page (see Figure 12-14 on page 12-30). Use the 1P indicator only with heading or detail output lines. It cannot be used to condition total or exception output lines or should not be used in an AND relationship with control level indicators.

- If certain error conditions occur, you might not want output operation processed. Use halt indicators to prevent the data that caused the error from being used (see Figure 12-15 on page 12-31).
- To condition certain output records on external conditions, use external indicators to condition those records.

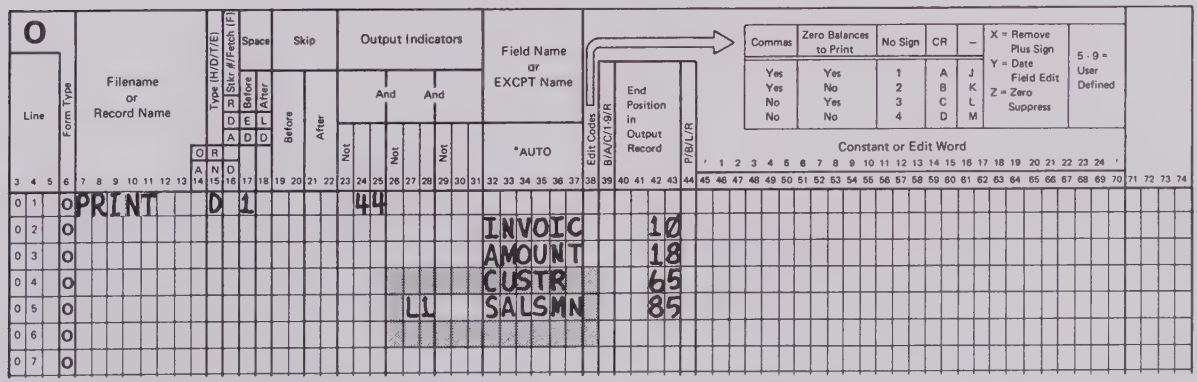
See "Printer Files" on page 6-17 in Chapter 6, "General File Considerations" for a discussion of the considerations that apply to assigning overflow indicators on the output specifications.

Output Specifications



One indicator is used to condition an entire line of printing. When 44 is on, the fields named INVOIC, AMOUNT, CUSTR, and SALSMN are all printed.

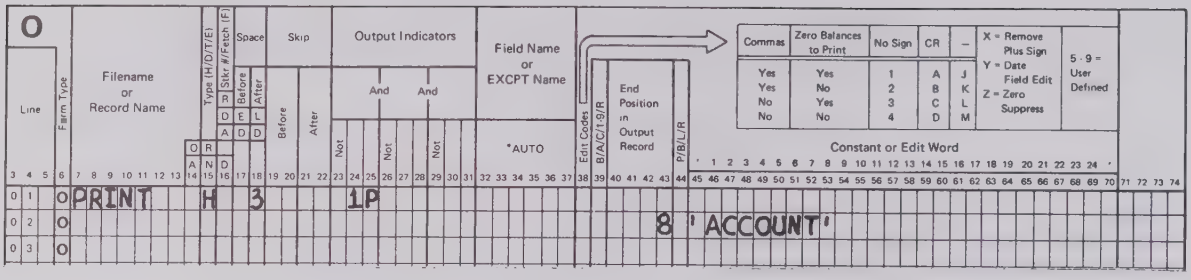
Output Specifications



A control level indicator is used to condition when one field should be printed. When indicator 44 is on, fields INVOIC, AMOUNT, and CUSTR are always printed. However, SALSMN is printed for the first record of a new control group only if 44 and L1 are on.

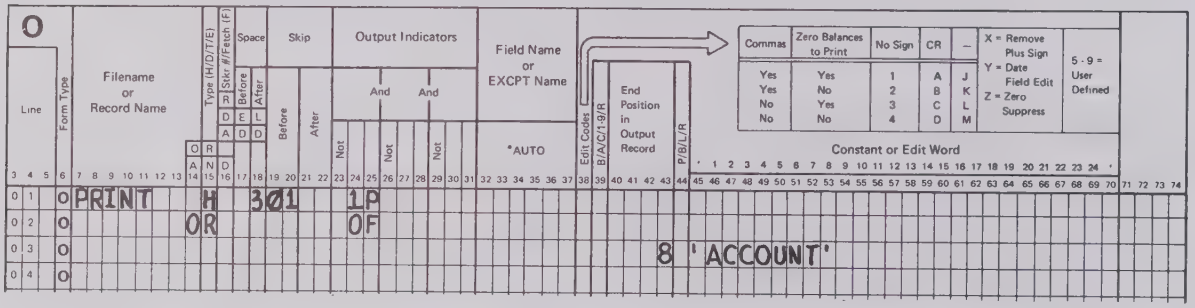
Figure 12-13. Output Indicators

Output Specifications



The 1P indicator is used when headings are to be printed on the first page only.

Output Specifications



The 1P indicator and an overflow indicator can be used to print headings on every page.

Figure 12-14. 1P Indicator

Input Specifications

```
I
Line  Form Type  Filename or Record Name  Sequence (O/A, R/N, O/D)  Number (1/N), E  Option (O), U, S  Record Identifying Indicator, ** or DS
External Field Name / Record Identification Codes: 1 (Position, Not (N), C/Z/D, Character) 2 (Position, Not (N), C/Z/D, Character) 3 (Position, Not (N), C/Z/D, Character)
Stacker Select  P/B/L/R  Field Location (From, To)  Data Structure (Occurs n Times, Length)  Decimal Positions  RPG Field Name
Control Level (L1-L9)  Matching Fields or Chaining Fields  Field Record Relation  Field Indicators (Plus, Minus, Zero or Blank)
3 4 5 | 6 | 7 8 9 10 11 12 13 | 14 15 16 | 17 | 18 | 19 20 | 21 22 23 24 | 25 | 26 27 | 28 29 30 31 | 32 | 33 34 | 35 36 37 38 | 39 | #0 | #1 | 42 | 43 | 44 45 46 47 | 48 49 50 51 | 52 | 53 54 55 56 57 58 | 59 60 | 61 62 | 63 64 | 65 66 | 67 68 | 69 70 | 71 72 73 74
01  I  CARD            AA   Ø1
02  I                                                                                                                                                1  3  FIELDAL1
03  I                                                                                                                                                4  8ØFIELDB                    H1
04  I
```

When an error condition (zero in FIELDB) is found, the halt indicator is set on.

Calculation Specifications

```
C
Line  Form Type  Control Level (L0-L9), LR, SR, AN/OR  Indicators (And, And; Not)  Factor 1  Operation  Factor 2
Result Field (Name, Length, Decimal Positions)  Half Adjust (H)  Resulting Indicators: Arithmetic (Plus, Minus, Zero); Compare (1>2, 1<2, 1=2); Lookup(Factor 2)is (High, Low, Equal)  Comments
3 4 5 | 6 | 7 8 | 9 10 11 12 13 14 | 15 16 17 | 18 19 20 21 22 23 24 25 26 27 | 28 29 30 31 32 | 33 34 35 36 37 38 39 40 41 42 | 43 44 45 46 47 48 | 49 50 51 | 52 | 53 54 | 55 56 | 57 58 | 59 | 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
01  C                  H1          GOTO  END
02  C                                 ≀
03  C                  Ø1             ≀
04  C                  Ø1             ≀
05  C                  Ø1             ≀
06  C                  END            TAG
07  C
```

≀ = Calculations

When H1 is on, all calculations are bypassed.

Output Specifications

```
O
Line  Form Type  Filename or Record Name  Type (H/D/T/E)  Stkr #/Fetch (F)  Space (Before, After)  Skip (Before, After)
Output Indicators (And, And; Not)  Field Name or EXCPT Name  Edit Codes  B/A/C/1-9/R  End Position in Output Record  P/B/L/R
Commas | Zero Balances to Print | No Sign | CR | – | X = Remove Plus Sign, Y = Date Field Edit, Z = Zero Suppress | 5-9 = User Defined
Constant or Edit Word
3 4 5 | 6 | 7 8 9 10 11 12 13 | 14 15 16 | 17 | 18 | 19 20 | 21 22 | 23 24 25 26 27 28 29 30 31 | 32 33 34 35 36 37 | 38 | 39 | 40 41 42 | 43 | 44 | 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 | 71 72 73 74
01  O  PRINT            H  Ø2Ø1        L1
02  O                                                                                                  5Ø  'HEADING'
03  O                  D  1Ø          Ø1NH1
04  O                                                  FIELDA                          5
05  O                                                  FIELDBZ                         15
```

FIELDA and FIELDB are printed only if H1 is not on. Therefore, if H1 is on, FIELDA and FIELDB are not printed. Use this general format when you do not want information that is in error to be printed.

Figure 12-15. Preventing Fields from Printing

Indicators Referred to as Data

An alternative method of referring to and manipulating RPG/400 indicators is provided by the RPG reserved words *IN and *INxx.

*IN

The array *IN is a predefined array of 99 one-position, character elements representing the indicators 01 through 99. The elements of the array can contain only the character values 0 (zero) or 1 (one).

The specification of the *IN array or the *IN,xx variable-index array element as a field in an input record, as a result field, or as factor 1 in a PARM operation defines indicators 01 through 99 for use in the program.

The operations or references valid for an array of single character elements are valid with the array *IN except that the array *IN cannot be specified as a subfield in a data structure, as a result field of a PARM operation, or in a SORTA operation.

*INxx

The field *INxx is a predefined one-position character field where xx represents any one of the RPG/400 indicators except 1P.

The specification of the *INxx field or the *IN,n fixed-index array element (where n = 1 - 99) as a field in an input record, as a result field, or as factor 1 in a PARM operation defines the corresponding indicator for use in the program.

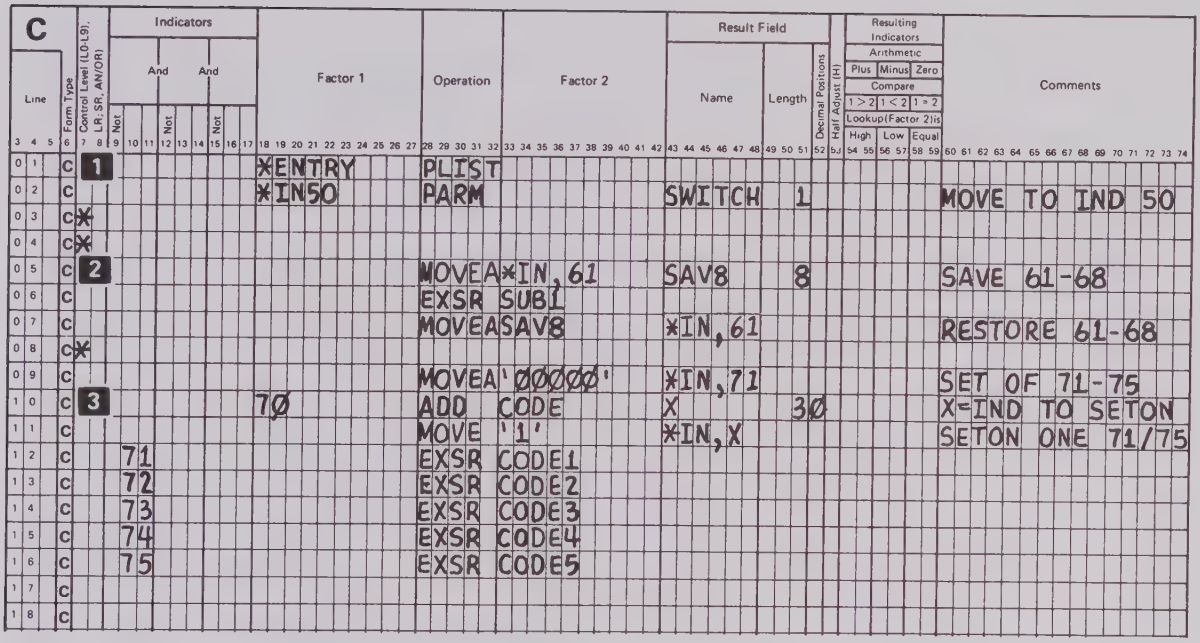
You can specify the field *INxx wherever a one-position character field is valid except that *INxx cannot be specified as a subfield in a data structure, as the result field of a PARM operation, or in a SORTA operation.

Additional Rules

Remember the following rules when you are working with the array *IN, the array element *IN,xx or the field *INxx:.

- Moving a character 0 (zero) to any of these fields sets the corresponding indicator off.
- Moving a character 1 (one) to any of these fields sets the corresponding indicator on.
- If the array *IN or the field *INxx is used to assign a value other than a character 0 (zero) or 1 (one) to an indicator, any subsequent normal RPG/400 indicator tests may yield undesirable results.

See Figure 12-16 on page 12-33 for some examples of indicators referred to as data.



Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: And / And (Not)	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus, Minus, Zero); Compare (1>2, 1<2, 1=2); Lookup (Factor 2) (High, Low, Equal)	Comments
01	C	1		*ENTRY	PLIST							
02	C			*IN50	PARM		SWITCH	1				MOVE TO IND 50
03	C	*										
04	C	*										
05	C	2			MOVEA *IN,61		SAV8	8				SAVE 61-68
06	C				EXSR SUBI							
07	C				MOVEA SAV8		*IN,61					RESTORE 61-68
08	C	*										
09	C				MOVEA '00000'		*IN,71					SET OF 71-75
10	C	3		70	ADD CODE		X	30				X=IND TO SETON
11	C				MOVE '1'		*IN,X					SETON ONE 71/75
12	C		71		EXSR CODE1							
13	C		72		EXSR CODE2							
14	C		73		EXSR CODE3							
15	C		74		EXSR CODE4							
16	C		75		EXSR CODE5							
17	C											
18	C											

1 When this program is called, a single parameter is passed to control some logic in the program. The parameter is used to set the value of indicator 50. The parameter must be passed with a character value of 1 or 0.

2 A subroutine uses indicators 61 through 68. Therefore, before the subroutine is processed, the status of these indicators as used in the mainline program is saved. (Assume that the indicators are set off in the beginning of the subroutine.) After the subroutine is processed, the indicators are returned to their original state.

3 A code field (CODE) contains a numeric value of 1 to 5. The field is used to set indicators 71 through 75. The five indicators are set off. Field X is calculated as 70 plus the CODE field. Field X is then used as the index into the array *IN. Different subroutines are then used based on the status of indicators 71 through 75.

Figure 12-16. Examples of Indicators Referred to as Data

Summary of Indicators

Figure 12-17 and Table 12-1 show a summary of where indicators are defined, what the valid entries are, where the indicators are used, and when the indicators are set on and off. Table 12-1 on page 12-35 indicates the primary condition that causes each type of indicator to be set on and set off by the RPG/400 program. Table 12-2 on page 12-36 shows the function key indicators and the corresponding function keys.

	User Defined	09-99	1P	H1-H9	L1-L9	LR	MR	OA-OG OV	U1-U8	KA-KN KP-KY	RT
DEFINED	Overflow indicator, file description specifications, positions 33-34	X									
	Record identifying indicator input specifications, positions 19-20	X		X	X	X			X		X
	Control level, input specifications, positions 59-60				X						
	Field level, input specifications, positions 65-70	X		X					X		X
	Resulting indicator, calculation specifications, positions 54-59	X		X	X	X		X**	X	X***	X
	RPG Defined										
	External indicator								X		
	Internal indicator		X			X	X				
USED	File conditioning, file description specifications, positions 71-72								X		
	File record relation, input specifications 63-64****	X		X	X		X		X		X
	Control level, calculation specifications, positions 7-8				X	X					
	Conditioning indicators, calculation specifications, positions 9-17	X		X	X	X	X	X	X	X	X
	Output indicators, output specifications, positions 23-31	X	X*	X	X	X	X	X	X	X	X

*The 1P indicator is allowed only on heading and detail lines.

**The overflow indicator must be defined on the file description specifications first.

***KA through KN, and KP through KY can be used as a resulting indicator only with the SETOF operation.

****Only a record identifying indicator from a main or OR record can be used to condition a control or match field. L1 through L9 cannot be used to condition a control or match field.

Figure 12-17. Indicator Entries and Uses

Table 12-1. When Indicators are Set On and Off

Type of Indicator	Set On	Set Off
Overflow	When printing on or spacing or skipping past the overflow line.	OA-OG, OV: After the following heading and detail lines are completed. 01-99: By the user.
Record identifying	When specified primary/secondary record has been read and before total calculations are processed; immediately after record is read from a full procedural file.	Before the next primary/secondary record is read during the next processing cycle.
Control level	When the value in a control field changes. All lower level indicators are also set on.	At end of following detail cycle.
Field indicator	By blank or zero in specified fields, by plus in specified field, or by minus in specified field.	Before this field status is to be tested the next time.
Resulting	When the calculation is processed and the condition that the indicator represents is met.	The next time a calculation is processed for which the same indicator is specified as a resulting indicator and the specified condition is not met.
Function key	When the corresponding function key is pressed for WORKSTN files.	By SETOF or move fields logic for a WORKSTN file.
External U1-U8	By CL command prior to beginning the program, or when used as a resulting or a field indicator.	By CL command prior to beginning the program, or when used as a resulting or when used as a resulting or a field indicator.
H1-H9	As specified by programmer.	When the continue option is selected as a response to a message, or by the programmer.
RT	As specified by programmer.	When the program is called again.
Internal Indicators		
1P	At beginning of processing before any input records are read.	Before the first record is read.
LR	After processing the last primary/secondary record of the last file or by the programmer.	At the beginning of processing, or by the programmer.
MR	If the match field contents of the record of a secondary file match the match field contents of a record in the primary file.	When all total calculations and output are completed for the last record of the matching group.

Table 12-2. Function Key Indicators and the Corresponding Function Keys

Function Key Indicator	Corresponding Function Key
KA	1
KB	2
KC	3
KD	4
KE	5
KF	6
KG	7
KH	8
KI	9
KJ	10
KK	11
KL	12
KM	13
KN	14
KP	15
KQ	16
KR	17
KS	18
KT	19
KU	20
KV	21
KW	22
KX	23
KY	24

Chapter 13. Using Arrays and Tables

An array is a systematic program-internal arrangement of data fields (array elements) with the same field length, data type (character or numeric), and number of decimal positions (if numeric). You can search an array sequentially for an identifiable array element, or you can refer to an array element by its position within the array. With some operations, you can refer to each of the array elements by using only the array name.

A table is also a systematic program-internal arrangement of data fields (table elements) with the same field length, data type (character or numeric), and number of decimal positions (if numeric). You search a table sequentially, using the LOKUP operation, to find a uniquely identifiable table element and any associated data. You cannot refer to table elements by their position within the table. Except in the LOKUP operation, the table name refers to the last table element found in a LOKUP operation. Unlike an array name, the table name does not refer to the entire set of table elements.

The next section describes how to code an array, how to specify the initial values of the array elements, how to change the values of an array, and the special considerations for using an array. The section after the next describes the same information for tables.

ARRAYS

There are three types of arrays:

- The *run-time array* is loaded by your program while it is running.
- The *compile-time array* is loaded when your program is created. It becomes a permanent part of your program.
- The *prerun-time array* is loaded from an array file when your program begins running, before any input, calculation, or output operations are processed.

The essentials of defining and loading an array are described for a run-time array. For defining and loading compile-time and prerun-time arrays you use these essentials and some additional specifications.

Array Name and Index

You refer to an entire array using the array name alone. You refer to the individual elements of an array using (1) the array name, followed by (2) a comma, followed by (3) an index (for example: AR,IND). The index indicates the position of the element within the array and is either a number or a field containing a number.

The following rules apply when you specify an array name and index:

- The array name must be a unique symbolic name.

- The array name with comma and index can be up to 6 characters long. If the array is only specified in factor 1 or factor 2 of calculation specifications, the array name with comma and index can be up to 10 characters long.
- The index is a numeric field with zero decimal positions, or a numeric constant.
- At run time, if your program refers to an array using an index with a value that is zero, negative, or greater than the number of elements in the array, then the error/exception routine takes control of your program.

Here are some examples of valid and invalid specifications of an array name and index:

- *Valid Array Names and Indexes:*

  `AR,1` This is the first element of array AR.

  `X,YY2` This is an element of array X. The index indicates which element of the array this is. `YY2` is the name of a field containing the index value.

- *Invalid Array Names and Indexes:*

  `BAL,XX1` The name including the comma has more than 6 characters. This name is only valid for factor 1 and factor 2 of calculation specifications. It is not a valid name for the result field because the result field is only six positions long.

  `AR,+1` The array name has an invalid signed index.

  `AR,0` The index value must be between 1 and the number of elements in the array, inclusive.

The Essential Array Specifications

Define an array on an extension specifications line. Here are the essential specifications for all arrays:

- Specify the array name in positions 27 through 32.
- Specify the number of entries in the array, right justified, in positions 36 through 39.
- Specify the length of an entry, right justified, in positions 40 through 42.
- If the array elements are numeric, specify the number of decimal positions in position 44.

Figure 13-1 on page 13-3 shows an example of the essential array specifications.

Coding a Run-Time Array

If you make no further specifications beyond the essential array specifications, you have defined a *run-time array*. Note that positions 33 through 35 must be blank for a run-time array.

Loading a Run-Time Array

You assign initial values for a run-time array through input or calculation specifications. You may also put data into other types of arrays this way.

For example, you may use the calculation specifications for the MOVE operation to put 0 in each element of an array (or in selected elements).

Using the input specifications, you may fill an array with the data from a file. The following sections provide more details on storing this data in the records of a file.

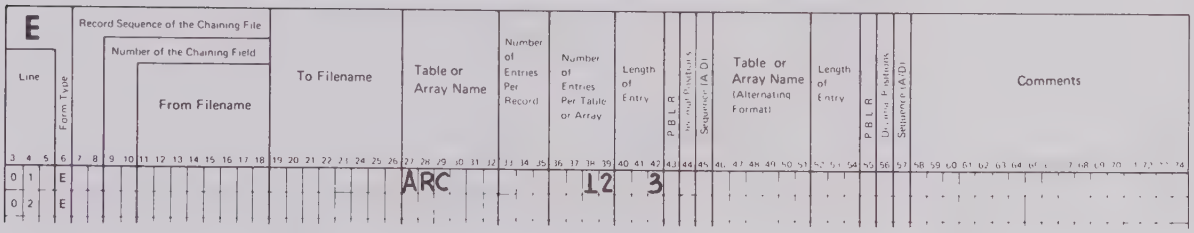


Figure 13-1. The Essential Array Specifications to Define a Run-Time Array

Array Information in One Record

If the array information is contained in one record, the information can occupy consecutive positions in the record or it can be scattered throughout the record.

If the array elements are consecutive on the input record, the array can be loaded with a single input specification. Figure 13-2 on page 13-4 shows the specifications for loading an array, INPARR, of six elements (12 characters each) from a single record from the file ARRFILE.

If the array elements are scattered throughout the record, they can be defined and loaded one at a time, with one element described on a specification line. Figure 13-3 on page 13-5 shows the specifications for loading an array, ARRX, of six elements with 12 characters each, from a single record from file ARRFILE; a blank separates each of the elements from the others.

When an array is located in a data structure, all of the elements of the array are consecutive in the data structure.

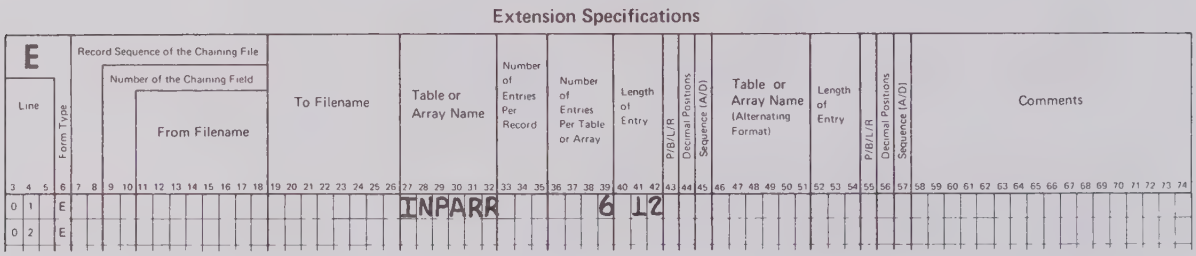
Array Information in More Than One Record

If the array information is in more than one record, you may use various methods to load the array. The method to use depends on the size of the array and whether or not the array elements are consecutive in the input records. Figure 13-4 on page 13-6 shows the array that results when array information is loaded from more than one input record. Each record identified by a 1 or 3 in position 1 contains 12 elements or array information. Records identified by a 2 in position 1 do not contain array information, although they appear in the same input file. Examples of loading and storing array information are found in "Examples of Using Arrays" on page 13-18. The RPG/400 program processes one record at a time. Therefore the

entire array is not processed until all the records containing the array information are read and the information is moved into the array fields. It may be necessary to suppress calculation and output operations until the entire array is read into the program.

Sequencing Run-Time Arrays

Run-time arrays are not sequence checked. If you process a SORTA (sort an array) operation, the array is sorted into the sequence specified on the extension specifications (A or D in position 45) defining the array. If the sequence is not specified, the array is sorted into ascending sequence. When the high (positions 54 and 55 of the calculation specifications) or low (positions 56 and 57 of the calculation specifications) indicators are used in the LOKUP operation, the array sequence must be specified.



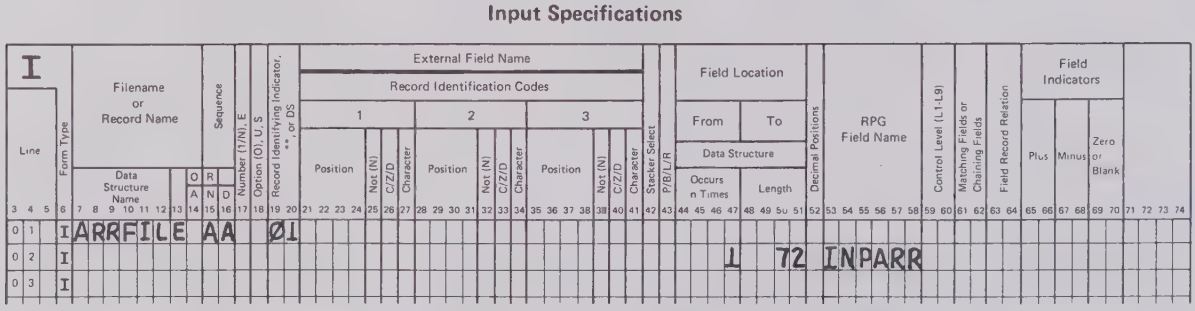


Figure 13-2. Defining a Run-Time Array with Consecutive Elements

Coding a Compile-Time Array

In addition to the essential array specifications, you must specify, in positions 33 through 35 of the extension specifications, how many arrays entries are in an array input record. See the specifications in Figure 13-5 on page 13-7.

Loading a Compile-Time Array

For a *compile-time array*, enter array input data into records in the program source member, following the source records for this program, and following the alternate collating sequence records and file translation records, if any. This data is loaded into the array when the program is compiled. Until the program is recompiled with

new data, the array will always initially have the same values each time you call the program.

Rules for Array Input Records

The rules for array input records are:

- The first array entry for each input record must begin in position 1.
- An entire record need not be filled with entries. If it is not, blanks or comments can be included after the entries (see Figure 13-5 on page 13-7). The unused entries in numeric arrays are filled with zeros; the unused entries in character arrays are filled with blanks.

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File / Number of the Chaining Field / From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E			ARRX		6	L2									
02	E															

Input Specifications

Line	Form Type	Filename or Record Name / Data Structure Name	Sequence A	Sequence N	Sequence D	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, **, or DS	External Field Name / Record Identification Codes 1 Position	1 Not (N)	1 C/Z/D	1 Character	2 Position	2 Not (N)	2 C/Z/D	2 Character	3 Position	3 Not (N)	3 C/Z/D	3 Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Data Structure Occurs n Times	Data Structure Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank
01	I	ARRFILE	AA					Ø1																										
02	I																						1	12			ARRX,1							
03	I																						14	25			ARRX,2							
04	I																						27	38			ARRX,3							
05	I																						40	51			ARRX,4							
06	I																						53	64			ARRX,5							
07	I																						66	77			ARRX,6							
08	I																																	

Figure 13-3. Defining a Run-Time Array with Scattered Elements

- Each input record except the last must contain the same number of entries. A record can contain one entry or as many entries as the record can hold with the exception of the last record. In the last record, unused entries must be blank and comments can be included after the unused entries. Comments in the last record must begin in the same position as comments in preceding records.
- Each entry must be contained entirely on one input record. An entry cannot be split between two records; therefore, the length of a single entry is limited to the maximum length of 80 characters (size of source record). If arrays are used and are described in alternating format, corresponding elements must be on the same input record; together they cannot exceed 80 characters.

- Arrays can be described separately or in alternating format. Alternating format means that the elements of one array are intermixed on the input record with elements of another array.

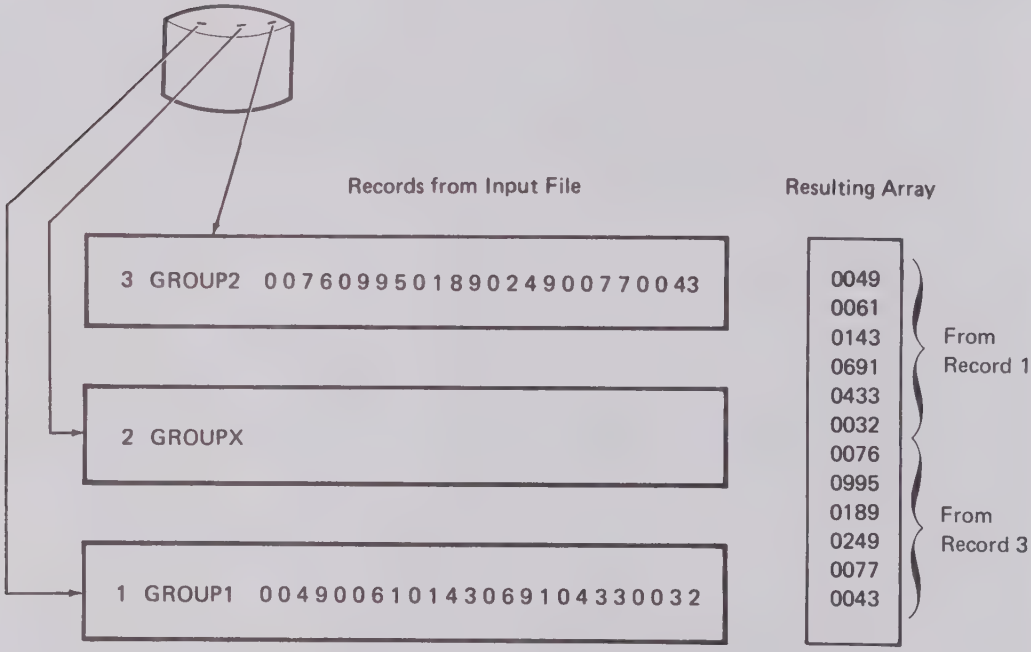


Figure 13-4. Loading an Array from Input Records

- All elements of an array must have the same characteristics (length, format and number of decimal positions, if numeric).
- A record with **b (b = blank) in positions 1 through 3 must precede the first input record of each array.
- Arrays are loaded in the order in which they are described in the extension specifications.
- Character array elements can contain a maximum of 80 positions. Numeric array elements can contain a maximum of 30 positions.
- Arrays can be in ascending, descending, or no sequence (A, D, or blank in positions 45 and 57 of the extension specifications).
- If L or R is specified in positions 43 and 55 of the extension specifications, the length specified for each element must include the sign (+ or -).

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File / Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E				ARC	L2	L2	3									
02	E																

48K
163
43J
640
44H
126
48A
473
49K
346
50B
125

The compile-time array, ARC, looks like this in the system.

The array entries in this record end at position 36. The remaining record positions can remain blank or can be used for comments as in this example.

```
48K16343J64044H12648A47349K34650B125          Array of Numbers
```

Figure 13-5. Array Input Record with Comments

Coding a Prerun-Time Array

On the extension specifications, in addition to the essential array specifications, specify the name of the file with the array input data, in positions 11 through 18. In positions 19 through 26, you may optionally specify the name of a file to which the array is written at the end of the program. If the file is a combined file (specified by a C in position 15 of the file description specifications), the names in 11 through 18 and 19 through 26 must be the same.

In position 43, specify a P if the array data is in packed format, B if the data is in binary format, L to indicate a sign on the left of a data element, or R to indicate a sign on the right of a data element. Otherwise, leave position 43 blank.

Specify a T in position 16 of the file description specifications for the file with the array input data.

Compare the coding of two prerun-time arrays, a compile-time array, and a run-time array in Figure 13-6 on page 13-8.

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File / Number of the Chaining Field / From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E			ARC	3	8	12		4		ARD	12		4		Compile-time arrays in alternating format
02	E	X														
03	E	DISKIN		ARE	12	250	5			A						Preexecution-time array
04	E	X														
05	E			ARI		10	10		0							Execution-time array
06	E	X														
07	E	DISKOUT	DISKOUT	ARH	12	250	5			A						Preexecution-time array specified as a combined file
08	E															
	E															
	E															

Figure 13-6. Extension Specifications for Four Types of Arrays

Figure 13-6 shows the extension specifications required for four types of arrays:

- Line 1 specifies two compile-time arrays in alternating format, ARC and ARD. Both arrays have eight elements (three elements per record). In both arrays, the length of each element is 12, with four decimal positions.
- Line 3 specifies a prerun-time array, ARE, to be read from file DISKIN. ARE has 250 character elements (12 elements per record). Each element is five positions long. The elements are arranged in ascending sequence.
- Line 5 specifies a run-time array, ARI, to be read from input records. ARI has 10 numeric elements, each 10 positions long with zero decimal positions.
- Line 7 specifies a prerun-time array, ARH, that is written back to the same file from which it is read. ARH has 250 character elements (12 elements per record). Each element is five positions long. The elements are arranged in ascending sequence.

For compile-time arrays, positions 19 through 26 can also contain the name of a file to which the array is to be written at end of job. For prerun-time arrays, positions 46 through 51 can also describe an array that is entered in alternating format with the array described in positions 27 through 32.

Loading a Prerun-Time Array

For a *prerun-time array*, enter array input data into a file. The file must be a sequential program described file. When you call a program, but before RPG/400 processes any input, calculation, or output operations, RPG/400 loads the array with initial values from the file. By modifying this file, you can alter the array's initial values on the next call to the program, without recompiling the program. The file is read in arrival sequence.

Defining More than One Array

There are three ways that you can define more than one array, they are:

- Defining two run-time arrays
- Mixing compile-time and pre-run time arrays
- Loading two compile-time or two prerun-time arrays in alternating format.

Two Run-Time Arrays

You can specify two run-time arrays on one extension specifications line by entering a second array name, length of entry, and decimal position in positions 46 through 57 of the first array's extension specifications.

Mixing Compile-Time and Prerun-Time Arrays

The specifications for compile-time and prerun-time arrays and tables can be intermixed in the extension specifications. The sequence in which arrays are specified in the extension specifications determines the order in which they are loaded at the start of the program.

Arrays in Alternating Format

You can load two compile-time arrays or two prerun-time arrays in *alternating format* by specifying a second array name, length of entry, and decimal position in positions 46 through 57 of the first array's extension specifications. The records for storing the data for such arrays have the first element of the first array followed by the first element of the second array, the second element of the first array followed by the second element of the second array, the third element of the first array followed by the third element of the second array, and so on. Corresponding elements must appear on the same record. The specification for the number of entries per record in positions 33 through 35 of the extension specifications indicates the number of corresponding pairs per record, each pair of elements counting as a single entry.

Figure 13-7 on page 13-10 shows two arrays, ARRA and ARRB, in alternating format.

ARRA (Part Number)	ARRB (Unit Cost)
345126	373
38A473	498
39K143	1297
40B125	93
41C023	3998
42D893	87
43K832	349
44H111	679
45P673	898
46C732	47587

Arrays ARRA and ARRB can be described as two separate array files or as one array file in alternating format.

The records for ARRA and ARRB look like the records below when described as two separate array files.

This record contains ARRA entries in positions 1 through 60.

This record contains ARRB entries in positions 1 through 50.

The records for ARRA and ARRB look like the records below when described as one array file in alternating format. The first record contains ARRA and ARRB entries in alternating format in positions 1 through 55. The second record contains ARRA and ARRB entries in alternating format in positions 1 through 55.

Figure 13-7. Arrays in Alternating and Nonalternating Format

Searching Arrays

The LOKUP operation can be used to search arrays. See the *Languages: RPG/400 Reference* for a description of the LOKUP operation.

Searching an Array without an Index

When searching an array without an index, use the status (on or off) of the resulting indicators to determine whether a particular element is present in the array. Searching an array without an index can be used for validity checking of input data to determine if a field is in a list of array elements. Generally, an equal LOKUP is requested.

In factor 1 in the calculation specifications, specify the search argument (data for which you want to find a match in the array named). Factor 1, the search argument, can be:

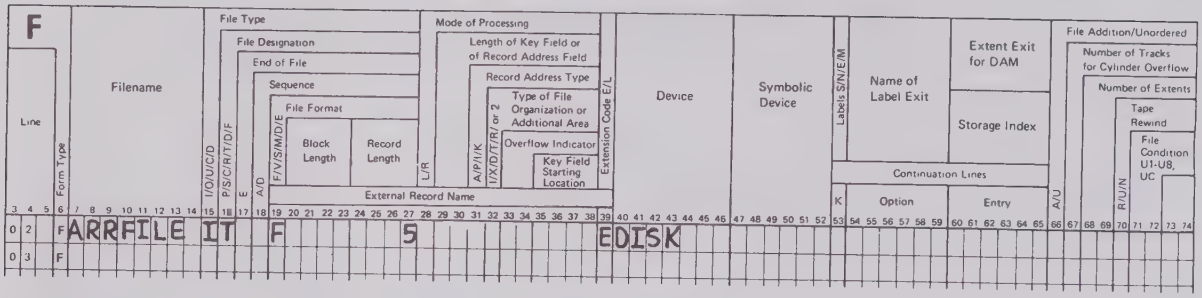
- A character or numeric literal
- A field name
- A data structure name
- An array element
- A table name.

Specify the operation code LOKUP in positions 28 through 32. In factor 2 specify the name of the array to be searched. At least one resulting indicator must be specified. Entries must not be made in both high and low for the same LOKUP operation. The resulting indicators must *not* be specified in high or low if the array is not in sequence (A or D in position 45 and/or position 57 of the extension specifications). Conditioning indicators (specified in positions 7 through 17) can also be used. The result field cannot be used.

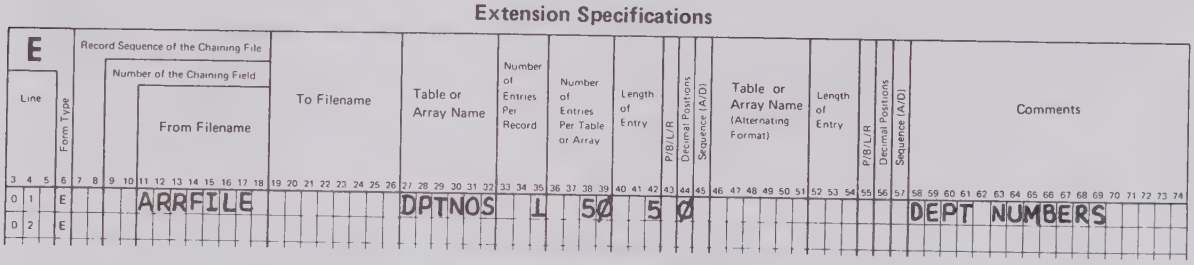
The search starts at the beginning of the array and ends at the end of the array or when the conditions of the lookup are satisfied. Whenever an array element is found that satisfies the type of search being made (equal, high, low), the resulting indicator is set on.

Figure 13-8 on page 13-12 shows an example of a LOKUP on an array without an index.

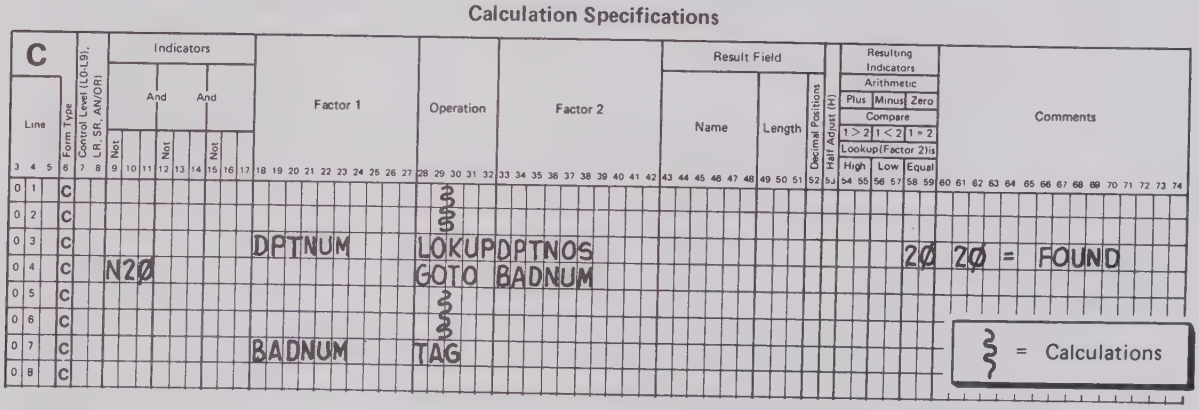
File Description Specifications



Extension Specifications



Calculation Specifications



ARRFILE, which contains department numbers, is defined in the file description specifications as an input file (I in position 15) with an array file designation (T in position 16). The file is program described (F in position 19), and each record is 5 positions in length (5 in position 27). The E in position 39 indicates that the file is further described in the extension specifications.

In the extension specifications, ARRFILE is defined as containing the array DPTNOS. The array contains 50 entries (positions 38 and 39). Each entry is 5 positions in length (position 42) with zero decimal positions (position 44). One department number can be contained in each record (1 in position 35). However, each record does not have to contain an entry. Any record that does not contain an entry is filled with zeros.

In the calculation specifications, the LOKUP operation (line 03) is processed and, if an element of DPTNOS equal to the search argument (DPTNUM) is found, indicator 20 is set on to indicate that the search was successful. If an element is not found that is equal to the search argument, indicator 20 is not set on and the GOTO statement conditioned by N20 (line 04) causes a branch to the BADNUM TAG (line 07).

Figure 13-8. LOKUP Operation for an Array without an Index

Searching an Array with an Index

To find out which element satisfies a LOKUP search, start the search at a particular element in the array. To do this type of search, make the entries in the calculation specifications as you would for an array without an index. However, in positions 33 through 42, enter the name of the array to be searched, followed by a comma and a numeric constant or the name of a numeric field (with zero decimal positions) containing the number of the element to be used. The numeric constant or numeric field provides the number of the element at which the search is to start. This numeric constant or field is called the index because it points to a certain element in the array. All other rules that apply to an array without an index apply to an array with an index.

Figure 13-9 on page 13-14 shows a LOKUP on an array with an index.

File Description Specifications

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	L/R	Mode of Processing	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels S/N/E/M	Name of Label Exit	Extent Exit for DAM	Storage Index	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8, UC	Continuation Lines	K	Option	Entry	A/U	R/U/N
02	F	ARRFILE	IT	F					5									EDISK															
03	F																																

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E			ARRFILE		DPTNOS		1	50		5	Ø	DPTDSC	20				DEPT NO/DESCRPTN
02	E																	

Calculation Specifications

Line	Form Type	Control Level (L0-L9), LR, SR, AN/OR	Indicators And	Indicators And	Factor 1	Operation	Factor 2	Result Field Name	Result Field Length	Result Field Decimal Positions	Half Adjust (H)	Resulting Indicators Arithmetic Plus	Arithmetic Minus	Arithmetic Zero	Compare 1>2	Compare 1<2	Compare 1=2	Lookup(Factor 2)is High	Lookup(Factor 2)is Low	Lookup(Factor 2)is Equal	Comments
01	C					Z-ADD	1	X	30												SET X FOR LOKUP
02	C				DPTNUM	LOKUP	DPTNOS,X													20	20 = FOUND
03	C		N20			GOTO	BADNUM														
04	C					MOVE	DPTDSC,X	DPTNAM	20												MOVE TO DPTNAM
05	C																				
06	C																				
07	C																				
08	C																				
09	C				BADNUM	TAG															
10	C																				

=Calculations

This example shows the same array of department numbers, DPTNOS, as Figure 13-8 on page 13-12. However, an alternating array of department descriptions, DPTDSC, is also defined in positions 46 through 51 of the extension specifications. Each element in DPTDSC is 20 positions in length (positions 53 and 54). Any record in DPTNOS that does not contain an entry is filled with zeros. Any record in DPTDSC that does not contain an entry is filled with blanks.

In the calculation specifications, the Z-ADD operation (line 03) begins the LOKUP (line 04) at the first element in DPTNOS. At the end of a successful LOKUP, when an element has been found that contains an entry equal to the search argument DPTNUM, indicator 20 is set on and the MOVE operation (line 06) places the department description, which corresponds to the department number, into DPTNAM. If an element is not found that is equal to the search argument, the GOTO operation conditioned by N20 causes a branch to the BADNUM TAG (line 09).

Figure 13-9. LOKUP Operation on an Array with an Index

Specifying Arrays

Arrays can be used in input, output, or calculation specifications (see "Examples of Using Arrays" on page 13-18 later in this chapter) after they are specified on the extension specifications.

Specifying an Array in Calculations

An entire array or individual elements in an array can be specified in calculation specifications. You can process individual elements like fields. Remember, if an array element is to be used as a result field, the array name with the comma and index cannot exceed 6 characters.

To specify an entire array, use only the array name, which can be used as factor 1, factor 2, or the result field. The following operations can be used with an array name: ADD, Z-ADD, SUB, Z-SUB, MULT, DIV, SQRT, MOVE, MOVEL, MOVEA, MLLZO, MLHZO, MHLZO, MHHZO, DEBUG, XFOOT, LOKUP, SORTA, PARM and DEFN.

Several other operations can be used with an array element only but not with the array name alone. These operations are BITON, BITOF, COMP, CABxx, TESTZ, TESTN, TESTB, MVR, DOUxx, DOWxx and IFxx.

When specified with an array name without an index, certain operations are repeated for each element in the array. These are ADD, Z-ADD, SUB, Z-SUB, MULT, DIV, SQRT, MOVE, MOVEL, MLLZO, MLHZO, MHLZO and MHHZO. The following rules apply to these operations when an array name without an index is specified:

- When factors 1 and 2 and the result field are arrays with the same number of elements, the operation uses the first element from every array, then the second element from every array until all elements in the arrays are processed. If the arrays do not have the same number of entries, the operation ends when the last element of the array with the fewest elements has been processed. When factor 1 is not specified for the ADD, SUB, MULT, and DIV operations, factor 1 is assumed to be the same as the result field.
- When one of the factors is a field, a literal, or a figurative constant and the other factor and the result field are arrays, the operation is done once for every element in the shorter array. The same field, literal, or figurative constant is used in all of the operations.
- The result field must always be an array.
- If an operation code uses factor 2 only (for example, Z-ADD, Z-SUB, SQRT, ADD, SUB, MULT, or DIV do not have factor 1 specified) and the result field is an array, the operation is done once for every element in the array. The same field or constant is used in all of the operations.
- Resulting indicators (positions 54 through 59) cannot be used because of the number of operations being processed.

Modifying Contents of Arrays

Arrays can be temporarily changed while the program is running when the array name is used as a result field in an arithmetic or move operation. The appropriate entry in the array is modified for the duration of the program. The next time the program runs, however, the array contains the original entries. Temporary changes can be made permanent if the input records are changed, or if the changed array is written at end of program.

Figure 13-10 shows the specifications for modifying the contents of arrays ARFL and ARLI.

Adding Entries to Arrays

Entries can be added to arrays before or during run of the program. The simplest way to add entries to an array is to write additional entries on the input records before the program runs. However, entries that are created by calculation operations or read from an input record can also be added during the running of a program.

Figure 13-11 on page 13-17 shows how entries are added to numeric arrays with the LOKUP and MOVE operations. Such entries are temporary unless they are written in array input records. If these entries are to become a permanent part of the array, they must be written in records and included with the other array file records.

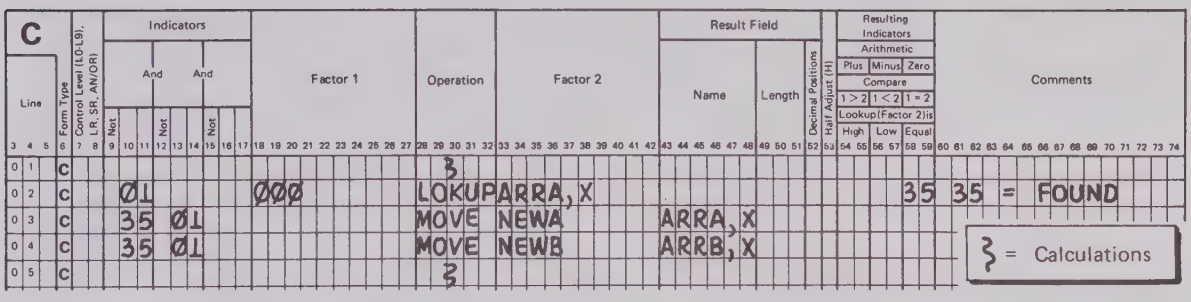
Calculation Specifications

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: And / And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Result Field: Length	Result Field: Decimal Positions	Half-Adjust (H)	Resulting Indicators: Arithmetic: Plus / Minus / Zero	Resulting Indicators: Compare: 1>2 / 1<2 / 1=2	Resulting Indicators: Lookup(Factor 2)is: High / Low / Equal	Comments
01	C				3									
02	C			25	LOKUP	ARFL,X								10 10 = FOUND
03	C		10		MOVE	500	ARLI,X							
04	C		10		MOVE	30	ARFL,X							
05	C				3									3 = Calculations

The element in ARFL,X that contains 25 to be changed to 30. The corresponding element in ARLI,X is to be changed to 500. The search argument is the numeric literal 25. The search starts at the array element specified in the index X. If the search is successful, the number of the array element is placed in the index field (X), and indicator 10 is set on. The new value in X can then be used to move 500 into the appropriate element in ARLI and to move 30 into the appropriate element in ARFL.

Figure 13-10. Changing Array Data with MOVE Operations

Calculation Specifications



The LOKUP operation is conditioned by indicator 01. Indicator 01 is set on when a record containing information in the fields NEWA and NEWB is read. These fields are to be added to the arrays ARRA and ARRB, respectively. To get the entry in the correct place in the array, a search is made to find the first empty array element. Unfilled entries in arrays are filled with zeros. Therefore, the search argument used is 000. When the first 000 entry is found, indicator 35 is set on, and the NEWA and NEWB fields are moved into the array elements ARRA,X and ARRB,X. These new elements become part of arrays ARRA and ARRB.

Figure 13-11. Adding Entries to Arrays

Array Output

Entire arrays can be written out under control of RPG/400 only at end of program when the LR indicator is on. To indicate that an entire array is to be written out, specify the name of the output file to be used in positions 19 through 26 of the extension specifications. This file must be described as a sequentially organized output or combined file in the file description specifications. If the file is a combined file and is externally described as a physical file, the information in the array at end of program replaces the information read into the array at the start of program. Logical files may give unpredictable results.

If an entire array is to be written to an output record (using output specifications), describe the array along with any other fields for the record:

- Positions 32 through 37 of the output specifications must contain the array name used in the extension specifications.
- Positions 40 through 43 of the output specifications must contain the record position where the last element of the array is to end. If an edit code is specified, the end position must include blank positions and any extensions due to the edit code (see "Editing Entire Arrays" listed next in this chapter).

Output indicators (positions 23 through 31) can be specified. Zero suppress (position 38), blank-after (position 39), and P = packed/B = binary and L = left/R = right (position 44) entries pertain to every element in the array.

Arrays that are loaded with data by means of input or calculation specifications cannot be written automatically at end of program.

Editing Entire Arrays

When editing is specified for an entire array, all elements of the array are edited. If different editing is required for various elements, refer to them individually.

When an edit code is specified for an entire array (position 38), two blanks are automatically inserted between elements in the array: that is, there are blanks to the left of every element in the array except the first. When an edit word is specified, the blanks are not inserted. The edit word must contain all the blanks to be inserted.

Examples of Using Arrays

Figure 13-12 on page 13-19 through Figure 13-18 on page 13-26 illustrate the following ways to use arrays:

- Building an array using fields as indexes (Figure 13-12 on page 13-19)
- Building an array using fixed indexes (Figure 13-13 on page 13-20)
- Calculating totals without arrays (Figure 13-14 on page 13-21)
- Calculating totals with arrays (Figure 13-15 on page 13-22)
- Using arrays to format field output (Figure 13-16 on page 13-23)
- Printing one array element per line (Figure 13-17 on page 13-25)
- Printing more than one array element per line (Figure 13-18 on page 13-26).

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E					AR		12	5									
02	E																	

Input Specifications

Line	Form Type	Filename or Record Name	Data Structure Name	Sequence O/A	Sequence R/N	Sequence D	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, ** or DS	External Field Name / Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Data Structure Occurs n Times	Data Structure Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank
01	I	FILE1		A	A				03	80			C1																						
02	I																							2	3				0X1						
03	I																							4	8				AR,X1						
04	I																							9	100				X2						
05	I																							11	15				AR,X2						
06	I																							16	17				0X3						
07	I																							18	22				AR,X3						
08	I																																		
09	I																																		
10	I																																		
11	I																							54	55				0X10						
12	I																							56	60				AR,X10						
13	I			B	B				04	80			C2																						
14	I																							2	3				0X1						
15	I																							4	8				AR,X1						
16	I																							9	100				X2						
17	I																							11	15				AR,X2						
18	I																																		

More Array Elements

This figure illustrates a method of loading an array using fields in input records as indexes. The array has 12 elements; each element is 5 positions long. The array could be defined with any number of elements (to a maximum of 99) without additional input specifications. To build an array using field indexes, assign different values to fields X1 through X10 on each input record type 03 and to fields X1 and X2 on each input record type 04. Succeeding type 03 records can then load 10 additional elements in array AR, up to the maximum defined in the array; each type 04 record can load two additional elements.

Blanks and other fields can appear on the input records because the array elements and their indexes are identified by the From and To entries.

To set up the array in this manner requires:

- A minimum of coding
- No calculations.

However, extra work is required to set up the indexing scheme for the input records.

Figure 13-12. Building an Array Using Input Fields as Indexes

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File / Number of the Chaining Field / From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E			AR1		30	5									
02	E															

RPG INPUT SPECIFICATIONS

Line	Form Type	Filename	Sequence (O A / R N / O)	Number (1/N)	Option (O), U, S	Record Identifying Indicator: or DS	Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank
01	I	FILE1	AA			03	100		C	1																				
02	I																				1	90		AR1						
03	I		BB			04	100		C	2																				
04	I																				1	5		AR1,19						
05	I																				6	10		AR1,20						
06	I																													
07	I																													
08	I																													
09	I																													
10	I																													
11	I																													

More Array Elements

This figure shows how eighteen 5-character elements of array AR1 are loaded with only two specification lines. On succeeding input specifications, the remaining elements of AR1 are loaded one after another until the array is full. Each additional element is coded on a separate line. Each new record requires a separate means of identification. For example, if another 03 record followed the first, the fields on the second record would overlay the fields read in from the first record. This method works well for small arrays.

Figure 13-13. Building an Array Using Fixed Indexes

Calculation Specifications

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: And Not	Indicators: And Not	Indicators: And Not	Factor 1	Operation	Factor 2	Result Field: Name	Result Field: Length	Result Field: Decimal Positions	Result Field: Half Adjust (H)	Resulting Indicators: Arithmetic: Plus	Resulting Indicators: Arithmetic: Minus	Resulting Indicators: Arithmetic: Zero	Resulting Indicators: Compare: 1>2	Resulting Indicators: Compare: 1<2	Resulting Indicators: Compare: 1=2	Resulting Indicators: Lookup (Factor 2): High	Resulting Indicators: Lookup (Factor 2): Low	Resulting Indicators: Lookup (Factor 2): Equal	Comments
01	C					FIELDA	ADD	L1A	L1A	62												
02	C					FIELDB	ADD	L1B	L1B	62												
03	C					FIELDC	ADD	L1C	L1C	62												
04	C					FIELDD	ADD	L1D	L1D	62												
05	C	L1				L1A	ADD	L2A	L2A	62												ADD TO L2 TOTLS
06	C	L1				L1B	ADD	L2B	L2B	62												
07	C	L1				L1C	ADD	L2C	L2C	62												
08	C	L1				L1D	ADD	L2D	L2D	62												
09	C	L2				L2A	ADD	L3A	L3A	62												ADD TO L3 TOTLS
10	C	L2				L2B	ADD	L3B	L3B	62												
11	C	L2				L2C	ADD	L3C	L3C	62												
12	C	L2				L2D	ADD	L3D	L3D	62												

Output Specifications

Line	Form Type	Filename or Record Name	Type (H/D/T/E)	Space: Before	Skip: Before	Output Indicators: And Not	Field Name or EXCPT Name	Edit Codes	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O		T	2		L1					
02	O						L1A	KB	15		
03	O						L1B	KB	30		
04	O						L1C	KB	45		
05	O						L1D	KB	60		
06	O		T	2		L2					
07	O						L2A	KB	15		
08	O						L2B	KB	30		
09	O						L2C	KB	45		
10	O						L2D	KB	60		
11	O		T	2		L3					
12	O						L3A	KB	15		
13	O						L3B	KB	30		
14	O						L3C	KB	45		
15	O						L3D	KB	60		

Commas: Yes / Yes / No / No
Zero Balances to Print: Yes / No / Yes / No
No Sign: 1 / 2 / 3 / 4
CR: A / B / C / D
–: J / K / L / M
X = Remove Plus Sign; Y = Date Field Edit; Z = Zero Suppress
5 9 = User Defined

The specifications in this figure tabulate three levels of totals. As they are read from input records, the fields FIELDA, FIELDB, FIELDC, and FIELDD are added to the first-level totals L1A, L1B, L1C, and L1D. These first-level totals are added at the time of an L1 control break to totals L2A, L2B, L2C, and L2D. Similarly, at an L2 control break, the second-level totals are added to third-level totals L3A, L3B, L3C, and L3D. In addition, as control breaks occur, L1, L2, and L3 total output is processed; and total fields are set to zeros after they are written to the output device. Figure 13-15 on page 13-22 shows the same tabulations done on arrays.

Figure 13-14. Calculating Totals without Arrays

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E					SL1		4	6		2							LL TOTALS
02	E					SL2		4	6		2							L2 TOTALS
03	E					SL3		4	6		2							L3 TOTALS
04	E																	

Calculation Specifications

Line	Form Type	Control Level (LO-L9, LR, SR, AN/OR)	Indicators: And Not	And Not	And Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic Plus	Minus	Zero	Compare 1>2	1<2	1=2	Lookup (Factor 2) is High	Low	Equal	Comments
01	C					FIELDA	ADD	SL1,1	SL1,1													
02	C					FIELDB	ADD	SL1,2	SL1,2													ADD FOR LL TOTL
03	C					FIELDC	ADD	SL1,3	SL1,3													
04	C					FIELDD	ADD	SL1,4	SL1,4													
05	C	L1				SL1	ADD	SL2	SL2													ADD FOR L2 TOTL
06	C	L2				SL2	ADD	SL3	SL3													ADD FOR L3 TOTL
07	C																					

Output Specifications

Line	Form Type	Filename or Record Name	Type (H/D/T/E)	Stkr #/Fetch (F)	Space: Before	Space: After	Skip: Before	Skip: After	Output Indicators: And Not	And Not	And Not	Field Name or EXCPT Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O		T		2	0			L1								
02	O											SL1		K	8	60	
03	O		T		2	0			L2								
04	O											SL2		K	8	60	
05	O		T		2	0			L3								
06	O											SL3		K	8	60	
07	O																

Commas: Yes, Yes, No, No — Zero Balances to Print: Yes, No, Yes, No — No Sign: 1, 2, 3, 4 — CR: A, B, C, D — –: J, K, L, M — X = Remove Plus Sign; Y = Date Field Edit; Z = Zero Suppress — 5-9 = User Defined

This figure is similar to Figure 13-14 on page 13-21 except that the three levels of totals are tabulated with arrays. Note the reduction in coding required to specify the functions. For example, line 5 of the calculation specifications fills the same function as lines 5 through 8 of the calculation specifications shown in Figure 13-14 on page 13-21. Similarly, the output specifications are reduced from 15 lines to 6. The method using arrays results in only two positions between array elements.

Figure 13-15. Calculating Totals with Arrays

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File / Number of the Chaining Field / From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E			ARA			4	5	0							
02	E			ARB			5	10								
03	E			ARC			6	4	2							
04	E															

Input Specifications

Line	Form Type	Filename or Record Name / Data Structure Name	Sequence (AND/OR)	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, ** or DS	External Field Name / Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location: From	Field Location: To	Data Structure: Occurs n Times	Data Structure: Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Field Indicators: Minus	Field Indicators: Zero or Blank
01	I	IN	AA			01	80		C																							
02	I	OR				02	80		C	1																						
03	I																					51	74		ARC							
04	I																					1	20		ARA			01				
05	I																					1	50		ARB			02				
06	I																															

Output Specifications

Line	Form Type	Filename or Record Name	Type (H/D/T/E)	Output Indicators: And	Field Name or EXCPT Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	OUT	D	01						
02	O		OR	02						
03	O				ARC			84		'0. &CR'
04	O			01	ARA,1	Z		89		
05	O			02	ARB,X1			100		
06	O									

Commas: Yes, Yes, No, No — Zero Balances to Print: Yes, No, Yes, No — No Sign: 1, 2, 3, 4 — CR: A, B, C, D — –: J, K, L, M — X = Remove Plus Sign; Y = Date Field Edit; Z = Zero Suppress — 5-9 = User Defined

Figure 13-16 (Part 1 of 2). Using Arrays to Format Field Output

This figure illustrates the use of three arrays to format field output. The arrays are defined as follows:

Array Name	Number of Elements	Element Length
ARA	4	5
ARB	5	10
ARC	6	4

Array ARA is contained in the input records with record identifying indicator 01, ARB in the records with record identifying indicator 02, and ARC in both types of records. Array ARC and the element of array ARA are to be included together in an output record as are arrays ARC and an element (identified by field X1) of array ARB. Every element in array ARC is edited according to the edit word '0b.bb&CR' (b = blank).

The contents of the arrays in the first two input records are as follows:

Record	Array	Array Contents
1	ARA	12345678901234567890
	ARC	01234567890123456789876N (note that N equals minus 5)
2	ARB	JOHNbDOEbbJOEbSMITHb LEEbMARXbbJIMbKNOTSb TIMbTYLERb
	ARC	(the same as record 1)

In the first output record, the location and contents of the arrays are as follows (b = blank):

Array	Location	Contents
ARA (first element)	85-89	12345
ARC	37-84	b1.23bbb45.67bbb 89.01bbb23.45bbb 67.89bbb87.65bCR

For the second output record assume that the content of field X1 is 4; the locations and contents of the arrays are as follows:

Array	Location	Contents
ARB (fourth element)	91-100	JIMbKNOTSb
ARC	37-84	b1.23bbb45.67bbb 89.01bbb23.45bbb 67.89bbb87.65bCR

Figure 13-16 (Part 2 of 2). Using Arrays to Format Field Output

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File / Number of the Chaining Field / From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E			AR2	5	21	15		0							
02	E															

Calculation Specifications

Line	Form Type	Control Level (L0-L9), LR, SR, AN/OR	Indicators: And / Not	And / Not	And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus / Minus / Zero); Compare (1>2 / 1<2 / 1=2); Lookup (Factor 2) (High / Low / Equal)	Comments
01	C	LR					DO	21	IN		30			DO 21 TIMES
02	C	LR					EXCPTTOTAL							
03	C	LR					END							
04	C													

Output Specifications

Line	Form Type	Filename or Record Name	Type (H/D/T/E): O / A / N / D	Space: Before / After	Skip: Before / After	Output Indicators: And / Not	Field Name or EXCPT Name	Edit Codes	End Position in Output Record	B/A/C/1-9/R	P/B/L/R	Constant or Edit Word
01	O	ARFILE	E	1			TOTAL					
02	O						AR2,IN		20			
03	O											

Edit Codes: Commas; Zero Balances to Print; No Sign; CR; –

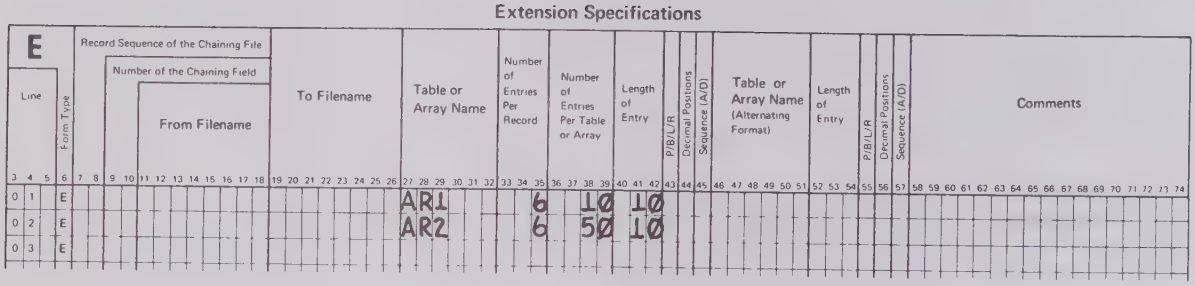
Commas	Zero Balances to Print	No Sign	CR	–
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign; Y = Date Field Edit; Z = Zero Suppress; 5-9 = User Defined

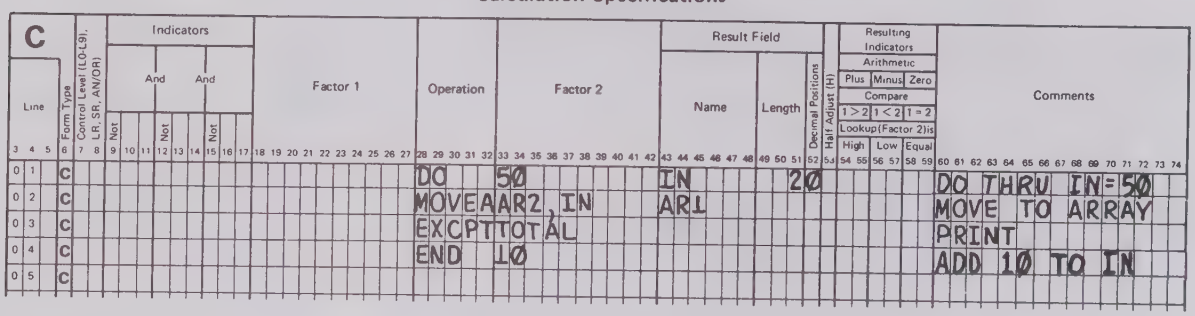
This figure shows a method of printing one array element per line on the printer output device.

Figure 13-17. Printing One Array Element per Line

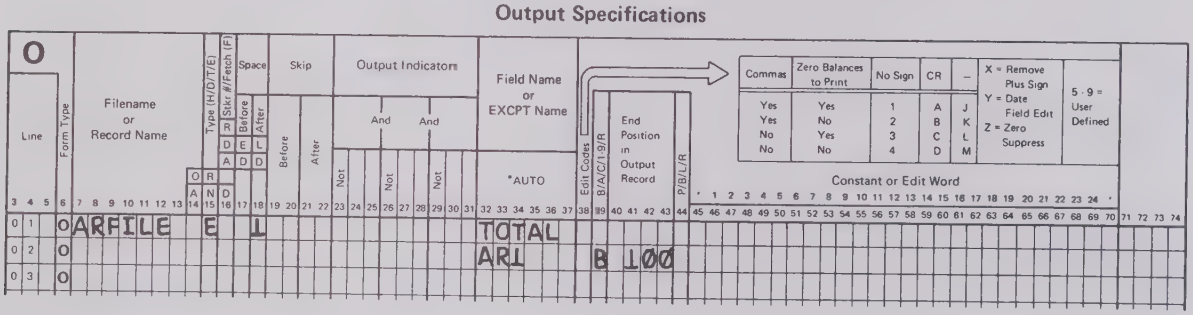
Extension Specifications



Calculation Specifications



Output Specifications



This figure shows a method of printing more than one array element per line on the printer output device.

Figure 13-18. Printing More than One Element per Line

TABLES

The explanation of arrays applies to tables except for the following deviations:

Activity	Deviation
Defining	A table name must be a unique symbolic name that begins with the letters TAB.
Loading	Tables can be loaded only at compilation time and prerun time.
Searching	The LOKUP operation can be used to search tables.

LOKUP with One Table

When a single table is searched, factor 1, factor 2, and at least one resulting indicator must be specified. Conditioning indicators (specified in positions 7 through 17) can also be used.

Whenever a table element is found that satisfies the type of search being made (equal, high, low), a copy of that table element is placed in an area in the system. Every time a search is successful, the newly found table element is placed in this area, replacing what was previously there. If the search is not successful, the contents of the area remain the same as they were before the unsuccessful search.

Before a first successful LOKUP, the area in the system reserved for the table element contains either the first element of the table or the element from any previous successful LOKUP.

Resulting indicators reflect the result of the search. If the indicator is on, reflecting a successful search, a copy of the element searched for is in the area.

LOKUP with Two Tables

When two tables are used in a search, only one is actually searched (see Figure 13-19 on page 13-28). When the search condition (high, low, equal) is satisfied, the corresponding elements from both tables are placed in their respective areas in the system and are made available for use.

Factor 1 must contain the search argument, and factor 2 must contain the name of the table to be searched. The result field must name the table from which data is also made available for use. A resulting indicator must also be used. Conditioning indicators can be specified in positions 7 through 17, if needed.

The two tables used should have the same number of entries. If the table that is searched contains more elements than the second table, it is possible to satisfy the search condition. However, there might not be an element in the second table that corresponds to the element found in the search table. Undesirable results can occur.

Note: If you specify a table name in an operation other than LOKUP before a successful LOKUP occurs, undesirable results can occur because the contents of the area referenced by the table name contain a previous value. The area is initialized to blanks or zeros prior to the first successful LOKUP.

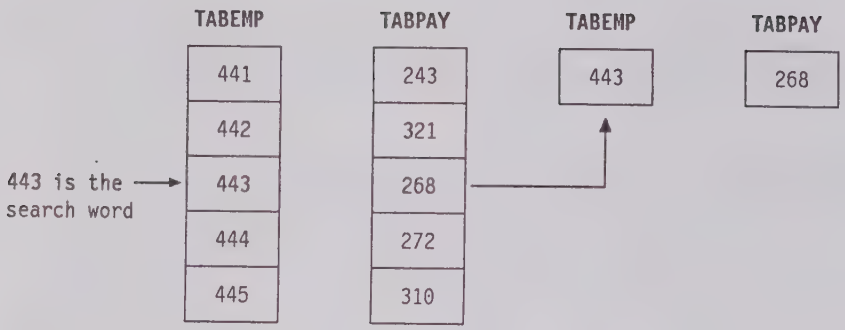
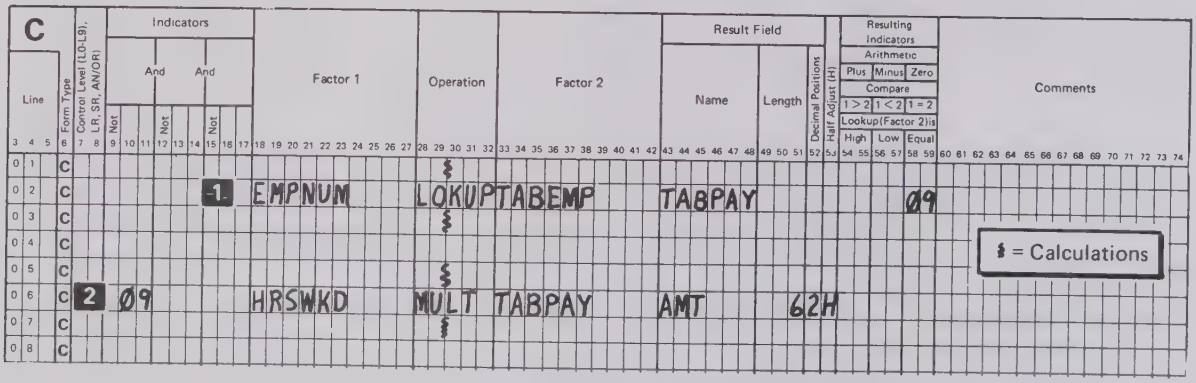


Figure 13-19. LOKUP Operation for Related Tables

Storage Areas

Tables TABEMP and TABPAY are read into storage. Assume an input record is read with 443 in the EMPNUM field. With 443 as the search argument, the table TABEMP can be searched for an equal entry. When the correct entry is found, the table item 443 is moved into the storage area for TABEMP. At the same time, the corresponding item 268 is moved into the storage area for TABPAY. The contents of the areas can now to be used in subsequent calculation operations by specifying the appropriate table name. The coding needed to process the LOKUP operation also shows how to refer to the contents of the storage area after a successful LOKUP operation.



1 The LOKUP operation searches TABEMP for an entry that is equal to the contents of the field named EMPNUM. If an equal entry is found in TABEMP, indicator 09 is set on, and the TABEMP entry and its related entry in TABPAY are moved into their storage areas.

2 If the conditions are such that indicator 09 is set on, the LOKUP operation multiplies the contents of the field named HRSWKD by the contents of the storage area for TABPAY. The storage area for TABPAY contains the element found during the last successful LOKUP operation involving TABPAY.

Specifying the Table Element Found in a LOKUP Operation

Whenever a table name is used in an operation other than LOKUP, the table name actually refers to the data retrieved by the last successful search. Therefore, when the table name is specified in this fashion, elements from a table can be used in calculation operations.

If the table is used as factor 1 in a LOKUP operation, the contents of the area in the system are used as the search argument. In this way an element from a table can itself become a search argument.

The table can also be used as the result field in operations other than the LOKUP operation. In this case the contents of the area in the system are changed by the calculation specification. The corresponding table element in the table in main storage is also changed. In this way the contents of the table can be modified by calculation operations (see Figure 13-20).

Calculation Specifications

| C | | | Indicators | | |

Chapter 14. Editing Numeric Fields

Editing provides a means of punctuating numeric fields, including the printing of currency symbols, commas, periods, and the sign status. It also provides for field sign movement from the rightmost digit to the end of the field, blanking zero fields, spacing in arrays, date field editing, and currency symbol or asterisk protection. A field can be edited by edit codes or edit words.

When you print fields that are not edited, the fields appear exactly as they are represented inside the computer. The following examples show why you may want to edit numeric output fields.

Type of Field	Field in the Computer	Printing of Unedited Field	Printing of Edited Field
Alphanumeric	JOHN T SMITH	JOHN T SMITH	JOHN T SMITH
Numeric (positive)	0047652	0047652	47652
Numeric (negative)	004765K	004765K	47652-

The unedited alphanumeric field and the unedited positive numeric field are easy to read when printed, but the unedited negative numeric field is confusing because it contains a K, which is not numeric. The K is a combination of the digit 2 and the negative sign for the field. They are combined so that one of the positions of the field does not have to be set aside for the sign. The combination is convenient for storing the field in the computer, but it makes the output hard to read. Therefore, numeric fields need to be edited before they are printed.

This chapter also contains information about using edit codes to edit externally described files.

Edit Codes

Edit codes are easier to use than edit words. Each of the edit codes edits in a different way according to a predefined pattern.

Edit codes are divided into three categories: simple (X, Y, Z), combination (1 through 4, A through D, J through M), and user-defined (5 through 9). You enter the edit code in position 38 of the output specifications for the field to be edited.

Simple Edit Codes

You can use simple edit to codes edit numeric fields without adding any punctuation. The simple edit codes and their functions are:

- The X edit code ensures a hexadecimal F sign for positive fields. However, because the system does this, you normally do not have to specify this code. Leading zeros are not suppressed. The X edit code does not modify negative numbers.
- The Y edit code is normally used to edit a three- to seven-digit date field. The Y edit code suppresses only the leftmost zero; however, if the field is seven positions long, the two leftmost positions are zero-suppressed. Figure 14-1 shows the editing for date fields. Positions 20 (date edit) and 21 (inverted print) of the control specification can be used to alter edit formats.
- The Z edit code removes the sign (plus or minus) from and suppresses the leading zeros of a numeric field. The decimal point is not placed in the field and is not printed.

		Control Specification					
				Contents of Position 21			
UPDATE	Edit Code	Contents of Position 19	Contents of Position 20	Blank	D	I	J
Jan 30, 1984	Y	Blank	Blank -	1/30/84 1-30-84	30/01/84 30-01-84	30.01.84 30-01-84	30.01.84 30-01-84
		M	Blank -	1/30/84 1-30-84	1/30/84 1-30-84	1/30/84 1-30-84	1/30/84 1-30-84
		D	Blank -	30.01.84 30-01-84	30.01.84 30-01-84	30.01.84 30-01-84	30.01.84 30-01-84
		Y	Blank -	84.01.03 84-01-30	84.01.30 84-01-30	84.01.30 84-01-30	84.01.30 84-01-30
Any character may be specified in position 20 as the separator character. For the purpose of this example, a dash has been used as the separator character.							

Figure 14-1. Edited Data Fields

Combination Edit Codes

The combination edit codes (1 through 4, A through D, J through M) punctuate a numeric field.

All of the edit codes suppress leading zeros to the left of the decimal notation except when J is specified in position 21 (inverted print) of the control specification (see the end of this section for further information on the J edit code). The decimal position of the source field determines whether and where a decimal point is printed. If decimal positions are specified for the source field and the zero balance is to be suppressed, the decimal notation prints *only* if the field is not zero. If a zero balance is not to be printed, a zero field prints as blanks.

When a zero balance is to be printed and the field is equal to zero, either of the following is printed:

- A decimal notation followed by n zeros, when n is the number of decimal places in the field
- A zero in the units position of a field if no decimal places are specified.

You can use a floating currency symbol or asterisk protection with any of the 12 combination edit codes. To specify a floating currency symbol, code the currency symbol in positions 45 through 47 of the output specifications, along with an edit code in position 38 for the field to be edited. The floating currency symbol appears to the left of the first significant digit. The floating currency symbol does not print on a zero balance when an edit code is used that suppresses the zero balance. (A dollar sign ($) is used as the currency symbol unless a currency symbol is specified in position 18 of the control specification.)

An asterisk constant coded in positions 45 through 47 of the output specifications ('*'), along with an edit code for the field to be edited causes an asterisk to be printed for each zero suppressed. A complete field of asterisks is printed on a zero balance source field.

Asterisk fill and the floating currency symbol *cannot* be used with the simple (X, Y, Z) or with the user-defined (5 through 9) edit codes.

A currency symbol can appear before the asterisk fill (fixed currency symbol). This requires the following coding:

1. Place an edit code in position 38.
2. Place a currency symbol constant one space before the beginning of the edited field.
3. Place '*' in positions 45 through 47 of the line containing the edit code.

When an edit code is used to print an entire array, two blanks precede each element of the array (except the first element).

One of the inverted print formats for output is a J entry in position 21 of the control specification. If you specify J, the use of commas and periods is reversed; that is, a period is used as the grouping separator character and a comma is used as the decimal notation. The zero-suppression character is in the second position (rather than the first) to the left of the decimal notation. This writes all zero balances and balances with zero values to the left of the comma with one leading zero (0,00 or 0,04). The J entry also overrides any edit codes that might suppress the leading zero. Remember that the decimal positions of the source field determine whether and where a decimal notation is printed.

Another inverted print format is an I in position 21 of the control specification. If you specify I, a period is used as the grouping separator character and a comma is used as the decimal notation.

Figure 14-2 on page 14-4 summarizes the functions of the combination edit codes. The codes edit the field in the format listed on the left. A negative field can be punctuated with no sign, CR, or a minus sign (-) as shown on the top of the figure.

	Negative Balance Indication		
Format of Edited Data	**No Sign**	CR	-
Prints with grouping separator characters, prints zero balance	1	A	J
Prints with grouping separator characters, zero balance suppressed	2	B	K
Prints without grouping separator characters, prints zero balance	3	C	L
Prints without grouping separator characters, zero balance suppressed	4	D	M

Figure 14-2. Summary of Combination Edit Codes

User-Defined Edit Codes

IBM has predefined edit codes 5 through 9. You can use them as they are, or you can delete them and create your own. For a description of the IBM-supplied edit codes, see "Edit Descriptions" in Chapter 6 of the *Programming: System Reference Summary*.

The user-defined edit codes allow you to handle common editing problems that would otherwise require the use of an edit word. Instead of the repetitive coding of the same edit word, a user-defined edit code can be used. These codes are system defined by the CL command CRTEDTD (Create Edit Description).

When you edit a field defined to have decimal places, be sure to use an edit word that has an editing mask for both the fractional and integer portions of the field. Remember that when a user-defined edit code is specified in a program, any system changes made to that user-defined edit code are not reflected until the program is recompiled. For further information on CRTEDTD, see the *Programming: Control Language Reference*.

Editing Considerations

Remember the following when you specify any of the edit codes:

- Edit fields of a nonprinter file with caution. If you do edit fields of a nonprinter file, be aware of the contents of the edited fields and the effects of any operations you do on them. For example, if you use the file as input, the fields written out with editing must be considered character fields, not numeric fields.
- Consideration should be given to data added by the edit operation. The amount of punctuation added increases the overall length of the output field. If these added characters are not considered, the output fields may overlap.

- The end position specified for output is the end position of the edited field. For example, if any of the edit codes J through M are specified, the end position is the position of the minus sign (or blank if the field is positive).

Summary of Edit Codes

Figure 14-3 on page 14-6 summarizes the edit codes and the options they provide. A simplified version of this table is printed above positions 45 through 70 on the output specifications. Table 14-1 on page 14-7 shows how fields look after they are edited.

Figure 14-4 on page 14-8 shows the effect that the different edit codes have on the same field with a specified end position for output.

				Entry in Column 21 of Control Specification			
Edit Code	Commas	Decimal Point	Sign for Negative Balance	D or Blank	I	J	Zero Suppress
1	Yes	Yes	No sign	.00 or 0	,00 or 0	0,00 or 0	Yes
2	Yes	Yes	No sign	Blanks	Blanks	Blanks	Yes
3		Yes	No sign	.00 or 0	,00 or 0	0,00 or 0	Yes
4		Yes	No sign	Blanks	Blanks	Blanks	Yes
5-9[1]							
A	Yes	Yes	CR	.00 or 0	0,00 or 0	Yes	
B	Yes	Yes	CR	Blanks	Blanks	Blanks	Yes
C		Yes	CR	.00 or 0	,00 or 0	0,00 or 0	Yes
D		Yes	CR	Blanks	Blanks	Blanks	Yes
J	Yes	Yes	-(minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
K	Yes	Yes	-(minus)	Blanks	Blanks	Blanks	Yes
L		Yes	-(minus)	.00 or 0	,00 or 0	0,00 or 0	Yes
M		Yes	-(minus)	Blanks	Blanks	Blanks	Yes
X[2]							
Y[3]							Yes
Z[4]							Yes

[1] These are the user-defined edit codes.
[2] The X edit code ensures a hexidecimal F sign for positive values. Because the system does this for you, normally you do not have to specify this code.
[3] The Y edit code suppresses the leftmost zero of a date field that is 3- to 6-digits long and it suppresses the two leftmost zeros of a field that is seven positions long. The Y edit code also inserts slashes(/) between athe month, day, and year according to the following pattern:
nn/n
nn/nn
nn/nn/n
nn/nn/nn
nnnn/nn/nn
[4] The Z edit code removes the sign (plus or minus) from a numeric field and suppresses leading zeros of a numeric field.

Figure 14-3. Edit Codes

Table 14-1. Examples of Edit Code Usage

Edit Codes	Positive Number-Two Decimal Positions	Positive Number-No Decimal Positions	Negative Number-Three Decimal Positions	Negative Number-No Decimal Positions	Zero Balance-Two Decimal Positions	Zero Balance-No Decimal Positions
Unedited	1234567	1234567	00012b[5]	00012b[5]	000000	000000
1	12,345.67	1,234,567	.120	120	.00	0
2	12,345.67	1,234,567	.120	120		
3	12345.67	1234567	.120	120	.00	0
4	12345.67	1234567	.120	120		
5-9[1]						
A	12,345.67	1,234,567	.120CR	120CR	.00	0
B	12.345.67	1,234,567	.120CR	120CR		
C	12345.67	1234567	.120CR	120CR	.00	0
D	12345.67	1234567	.120CR	120CR		
J	12,345.67	1,234,567	.120-	120-	.00	0
K	12,345,67	1,234,567	.120-	120-		
L	12345.67	1234567	.120-	120-	.00	0
M	12345.67	1234567	.120-	120-		
X[2]	1234567	1234567	00012b[5]	00012b[5]	000000	000000
Y[3]			0/01/20	0/01/20	0/00/00	0/00/00
Z[4]	1234567	1234567	120	120		

[1] These are the user-defined edit codes.

[2] The X edit code ensures a hex F sign for positive values. Because the system does this for you, normally you do not have to specify this code.

[3] The Y edit code suppresses the leftmost zero of a date field that is 3- to 6-digits long, and it suppresses the two leftmost zeros of a field that is seven positions long. The Y edit code also inserts slashes (/) between the month, day, and year according to the following pattern:

```
 nn/n
 nn/nn
 nn/nn/n
 nn/nn/nn
nnn/nn/nn
```

[4] The Z edit code removes the sign (plus or minus) from a numeric field and suppresses leading zeros of a numeric field.

[5] The b represents a blank. This may occur if a negative zero does not correspond to a printable character.

	Negative Number, 2 Decimal Positions. End Position Specified as 10.								
	Output Print Positions								
Edit Code	3	4	5	6	7	8	9	10	11
Unedited				0	0	4	1	K[1]	
1					4	.	1	2	
2					4	.	1	2	
3					4	.	1	2	
4					4	.	1	2	
5-9[2]									
A			4	.	1	2	C	R	
B			4	.	1	2	C	R	
C			4	.	1	2	C	R	
D			4	.	1	2	C	R	
J				4	.	1	2	-	
K				4	.	1	2	-	
L				4	.	1	2	-	
M				4	.	1	2	-	
X				0	0	4	1	K[1]	
Y			0	/	4	1	/	2	
Z						4	1	2	

[1]K represents a negative 2.
[2]These are user-defined edit codes.

Figure 14-4. Effect of Edit Codes on End Position

Edit Words

If you have editing requirements that cannot be met by using the edit codes described above, you can use an edit word. An edit word allows you to directly specify:

- Blank spaces
- Commas and decimal points, and their position
- Suppression of unwanted zeros
- Leading asterisks

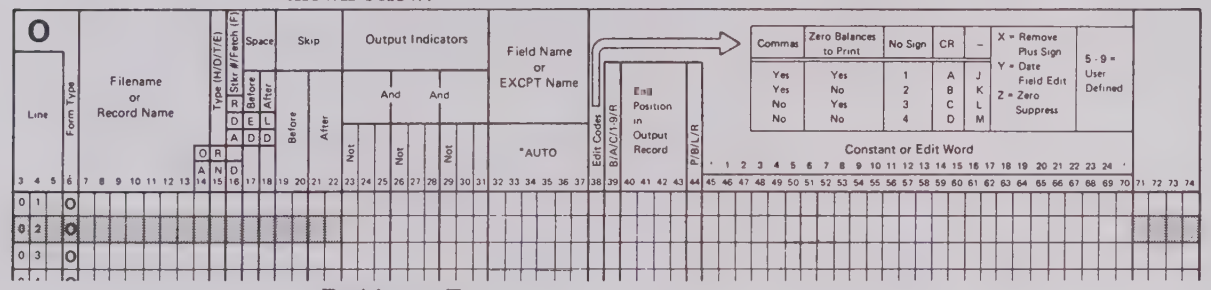
- The currency symbol, and its position
- Addition of constant characters
- Output of the negative sign, or CR, as a negative indicator.

The edit word is used as a template, which the system applies to the source data to produce the output.

You cannot use an edit word on fields having more than 24 characters. However, if the file is externally-defined, you can use DDS to edit fields over 24 characters long.

How to Code an Edit Word

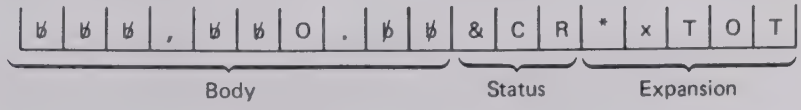
To use an edit word, code the unshaded portion of the output specifications as shown below:



Position	Entry
23-31	Can contain conditioning indicators.
32-37	Contains the name of the numeric field from which the data that is to be edited is taken.
38	*Edit code.* Must be blank, if you are using an edit word to edit the source data.
39	A "B" in this position indicates that the source data is to be set to zero or blanks after it has been edited and output. Otherwise the source data remains unchanged.
40-43	Identifies the end (rightmost) position of the field in the output record.
45-70	*Edit word.* Can be up to 24 characters long and must be enclosed by apostrophes. Enter the leading apostrophe into column 45. The edit word, itself, must begin in column 46.

Parts of an Edit Word

An edit word (coded into positions 45 to 70 of the output specifications) consists of three parts: the body, the status, and the expansion. The following shows the three parts of an edit word:



The *body* is the space for the digits transferred from the source data field to the output record. The body begins at the leftmost position of the edit word. The

number of blanks (plus one zero or an asterisk) in the edit word body must be equal to or greater than the number of digits of the source data field to be edited. The body ends with the rightmost character that can be replaced by a digit.

The *status* defines a space to allow for a negative indicator, either the two letters CR or a minus sign (-). The negative indicator specified is output only if the source data is negative. All characters in the edit word between the last replaceable character (blank, zero suppression character) and the negative indicator are also output with the negative indicator only if the source data is negative; if the source data is positive, these status positions are replaced by blanks. Edit words without the CR or - indicators have no status positions.

The status must be entered after the last blank in the edit word. If more than one CR followed the last blank, only the first CR is treated as a status; the remaining CRs are treated as constants. For the minus sign to be considered as a status, it must be the last character in the edit word.

The *expansion* is a series of ampersands and constant characters entered after the status. Ampersands are replaced by blank spaces in the output; constants are output as is. If status is not specified, the expansion follows the body.

Forming the Body of an Edit Word

The following characters have special meanings when used in the body of an edit word:

Blank: Is replaced with the character from the corresponding position of the source data field specified by the field name in positions 32 through 37 of the output specifications. A blank position is referred to as a digit position.

Decimals and Commas: Decimals and commas are in the same relative position in the edited output field as they are in the edit word unless they appear to the left of the first significant digit in the edit word. In that case, they are blanked out or replaced by an asterisk.

In the following examples below, all the leading zeros will be suppressed (default) and the decimal point will not print unless there is a significant digit to its left.

Edit Word	Source Data	Appears in Output Record as:
C ,	0000072	bbbbb72
C . ,	0000000012	bbbbbbbbb12
C . ,	0000000123	bbbbbbb1.23

Zeros: The first zero in the body of the edit word is interpreted as an end-zero-suppression character. This zero is placed where zero suppression is to end. Subsequent zeros put into the edit word are treated as constants (see "Constants" below).

Any leading zeros in the source data are suppressed up to and including the position of the end-zero-suppression character. Significant digits that would appear in the end-zero-suppression character position, or to the left of it, are output.

Edit Word	Source Data	Appears in Output Record as:
Z0000000Z	00000004	␣␣␣␣00000 4
Z0000000Z	012345	␣␣␣␣012345
Z0000000Z	012345678	␣␣12345678

If the leading zeros include, or extend to the right of, the end-zero-suppression character position, that position is replaced with a blank. This means that if you wish the same number of leading zeros to appear in the output as exist in the source data, the edit word body must be wider than the source data.

Edit Word	Source Data	Appears in Output Record as:
Z00Z	0156	␣156
Z00Z	0156	␣0156

Constants (including commas and decimal point) that are placed to the right of the end-zero-suppression character are output, even if there is no source data. Constants to the left of the end-zero-suppression character are only output if the source data has significant digits that would be placed to the left of these constants.

Edit Word	Source Data	Appears in Output Record as:
Z0000.0Z	000000001	␣␣␣␣␣␣␣.01
Z0000.0Z	000000000	␣␣␣␣␣␣␣.00
Z,00.0Z	00000012	␣␣␣␣␣0.12
Z,00.0Z	00000123	␣␣␣␣␣1.23
Z0,00.0Z	00000123	␣␣0,001.23

Asterisk: The first asterisk in the body of an edit word also ends zero suppression. Subsequent asterisks put into the edit word are treated as constants (see "Constants" below). Any zeros in the edit word following this asterisk are also treated as constants. There can be only one end-zero-suppression character in an edit word, and that character is the first asterisk *or* the first zero in the edit word.

If an asterisk is used as an end-zero-suppression character, all leading zeros that are suppressed are replaced with asterisks in the output. Otherwise, the asterisk suppresses leading zeros in the same way as described above for "Zeros".

Edit Word	Source Data	Appears in Output Record as:
Z*****.Z	000000123	*000001.23
Z****.Z	000000000	******0.00
Z****.**Z	00056342	***563.42**

Note that leading zeros appearing after the asterisk position are output as leading zeros. Only the suppressed leading zeros, including the one in the asterisk position, are replaced by asterisks.

Currency Symbol: A currency symbol followed directly by a first zero in the edit word (end-zero-suppression character) is said to float. All leading zeros are suppressed in the output and the currency symbol appears in the output immediately to the left of the most significant digit.

Edit Word	Source Data	Appears in Output Record as:
`C  J  J  $Ø.  J`	000000012	␣␣␣␣␣␣␣␣␣$.12
`C  J  J  $Ø.  J`	000123456	␣␣␣␣$1,234.56

If the currency symbol is put into the first position of the edit word, then it will always appear in that position in the output. This is called a fixed currency symbol.

Edit Word	Source Data	Appears in Output Record as:
`C$  J  J  Ø.  J`	000123456	$␣␣␣␣1,234.56
`C$  J  J  ØØ  Ø.  J`	000000000	$␣␣␣␣␣␣␣␣00.00
`C$  J  J  *  .  J`	000123456	$****1,234.56

A currency symbol anywhere else in the edit word and not immediately followed by a zero end-suppression-character is treated as a constant (see "Constants" below).

Ampersand: Causes a blank in the edited field. The example below might be used to edit a telephone number. Note that the zero in the first position is required to print the constant AREA.

Edit Word	Source Data	Appears in Output Record as:
`CØAREA&  &NO.&  -  J`	4165551212	␣AREA␣416␣NO.␣555-1212

Constants: All other characters entered into the body of the edit word are treated as constants. If the source data is such that the output places significant digits or leading zeros to the left of any constant, then that constant appears in the output. Otherwise, the constant is suppressed in the output. Commas and the decimal point follow the same rules as for constants. Notice in the examples below, that the presence of the end-zero-suppression character as well as the number of significant digits in the source data, influence the output of constants.

The following edit words could be used to print checks. Note that the second asterisk is treated as a constant, and that, in the third example, the constants preceding the first significant digit are not output.

Edit Word	Source Data	Appears in Output Record as:
`C$  **DOLLARS&  &CTS J`	000012345	$****123*DOLLARS␣45␣CTS
`C$  **DOLLARS&  &CTS J`	000000006	$*******DOLLARS␣06␣CTS
`C$  &DOLLARS&  &CTS J`	000000006	$␣␣␣␣␣␣␣␣␣

A date could be printed by using either edit word:

Edit Word	Source Data	Appears in Output Record as:
C / / b	010385	b1/03/85
C 0 / / b	010386	b01/03/86

Note that any zeros or asterisks following the first occurrence of either are treated as constants. The same is true for - and CR:.

Edit Word	Source Data	Appears in Output Record as:
C 0 . 0 0 0 b	01234	b12.34000
C * . 0 0 0 b	01234	*12.34000

Forming the Status of an Edit Word

The following characters have special meanings when used in the status of an edit word:

Ampersand: Causes a blank in the edited output field. An ampersand cannot be placed in the edited output field.

CR or minus symbol: If the sign in the edited output is plus (+), these positions are blanked out. If the sign in the edited output field is minus (−), these positions remain undisturbed.

The following example adds a negative value indication. The minus sign will print only when the value in the field is negative. A CR symbol fills the same function as a minus sign.

Edit Word	Source Data	Appears in Output Record as:
C . - b	000000123-	bbbbbb1.23-
C . - b	000000123	bbbbbb1.23b

Constants between the last replaceable character and the - or CR symbol will print only if the field is negative; otherwise, blanks will print in these positions. Note the use of ampersands to represent blanks:

Edit Word	Source Data	Appears in Output Record as:
C b b 0 . & 3 0 & D A Y & C R b	000000123-	bbbbbbbb1.23b30bDAYbCR
C b b 0 . & 3 0 & D A Y & C R b	000000123	bbbbbbbbb1.23bbbbbbbbbb

Formatting the Expansion of an Edit Word

The characters in the expansion portion of an edit word are always written. The expansion cannot contain blanks. If a blank is required in the edited output field, specify an ampersand in the body of the edit word.

Constants may be added to print on every line:

Edit Word	Source Data	Appears in Output Record as:
' , Ø. &CR&NET'	000123-	ØØØ1.23Ø CRØNET
' , Ø. &CR&NET'	000123	ØØØ1.23ØØØNET

Note that the CR in the middle of a word may be detected as a negative field value indication. If a word such as SECRET is required, use the coding in the example below.

Edit Word	Source Data	Appears in Output Record as:
' Ø. &SECRET'	12345-	123.45Ø SECRET
' Ø. &SECRET'	12345	123.45ØØØSECRET
' Ø. &CR&&SECRET'	12345	123.45ØØØØSECRET

Summary of Coding Rules for Edit Words

The following rules apply to edit words:

- Position 38 (edit codes) must be blank.
- Positions 32 through 37 (field name) must contain the name of a numeric field.
- An edit word must be enclosed in apostrophes. Enter the leading apostrophe in position 45. The edit word itself must begin in position 46.
- The edit word can contain more digit positions (blanks plus the initial zero or asterisk) than the field to be edited, but must not contain less. If there are more digit positions in the edit word than there are digits in the field to be edited, leading zeros are added to the field before editing.
- If leading zeros from the source data are desired, the edit word must contain one more position than the field to be edited, and a zero must be placed in the high-order position of the edit word.
- When the floating currency symbol is used, the sum of the number of blanks and the zero-suppression stop character (digit positions) contained in the edit word must be equal to or greater than the number of positions in the field to be edited. The floating currency symbol is not counted as a digit position.
- Any zeros or asterisks following the leftmost zero or asterisk are treated as constants; they are not replaceable characters.

Formatting Edit Words

A printer spacing chart can help you format edit words. Figure 14-5 on page 14-16 shows how this chart can be used to format an output line.

If you need to show a negative number, include a sign in the edit word. Use either the letters CR or the minus sign (-). These print only for a negative number; however, you must consider the character positions they require when you enter the end position of the field on the output specifications (see Figure 14-5 on page 14-16).

To use a minus sign to indicate a negative number, leave a space between the number and the negative sign, and place an ampersand (&) in the edit word before the minus sign. PERCPL then prints as 25b-.

If you want to print a currency symbol, you also indicate this in your edit word. Figure 14-6 on page 14-17 shows examples of printing a currency symbol.

Editing Externally Described Files

Edit codes must be specified in data description specifications (DDS), instead of RPG/400, to edit output for externally described files. See the *Programming: Data Description Specifications Reference* for information on how to specify edit codes in the data description specifications. However, if an externally described file, which has an edit code specified, is to be written out as a program described output file, you must specify editing in the output specifications. In this case, RPG/400 ignores any edit codes in the data description specifications.

Unedited Data

```
Item number        000241
Item cost          02000
Selling price      02200
% profit or loss   25
```

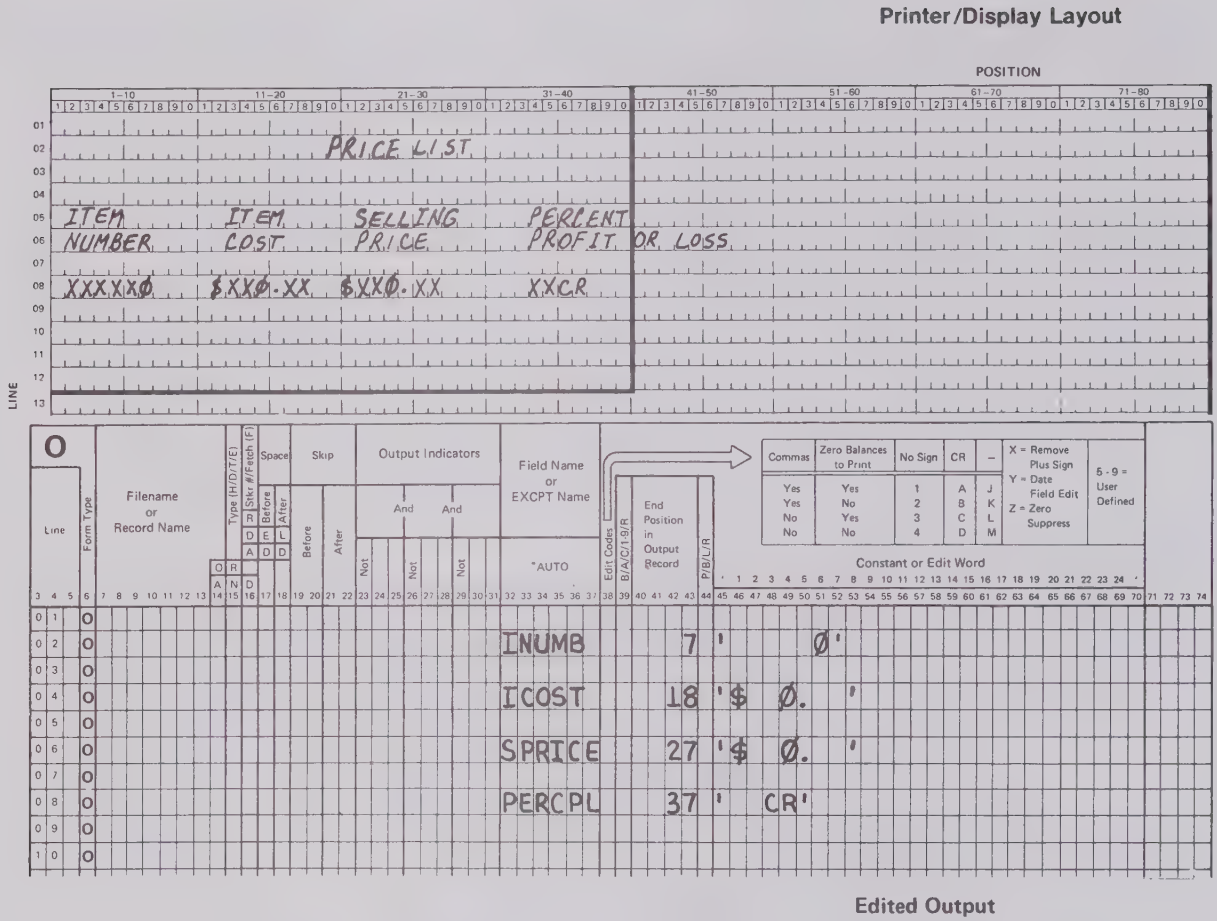


Figure 14-5. Using a Printer Spacing Chart to Format Data

The Xs and zeros on the printer spacing chart show field positions. A zero indicates where zero suppression ends. An X indicates that any number can appear in the position. Use blanks in place of the Xs when writing the edit words on the output specifications.

The specifications for the field PERCPL show how to print CR for a negative balance. Assume that the field PERCPL contains the negative data 2N (which is -25). The printed output will be 25CR. If PERCPL is positive, CR does not print and the same field appears as 25.

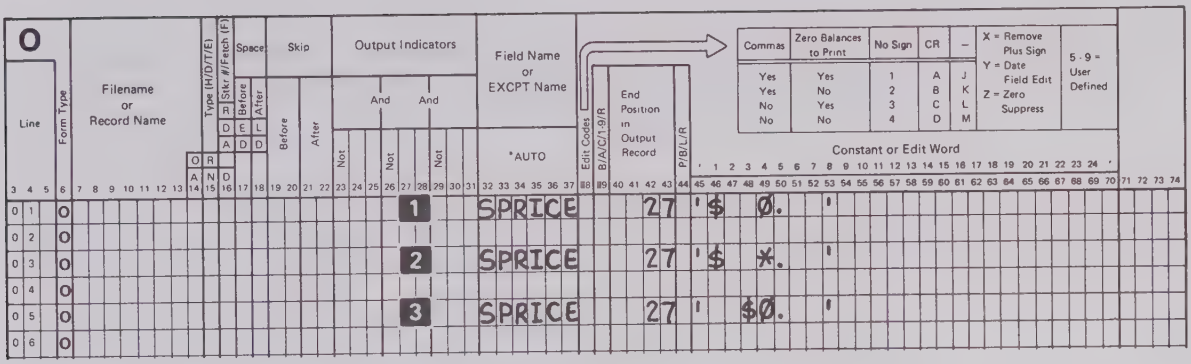


Figure 14-6. Different Edit Words Used on the Same Field

1 To print a currency symbol at the left of the field, put the currency symbol next to the first apostrophe, and then put in the necessary blanks and punctuation. A currency symbol in this position is called a fixed currency symbol. The SPRICE field can look like any of the following (N stands for any number):

```
$NNN.NN
$ NN.NN
$  N.NN
$   .NN
```

2 To avoid blanks between the currency symbol and the first digit when zero suppression occurs, indicate asterisk (*) fill. Instead of using 0 to indicate zero suppression, use the asterisk (*) to indicate that all extra spaces should be filled with asterisks. The SPRICE field can look like any of the following (N stands for any number):

```
$NNN.NN
$*NN.NN
$**N.NN
$***.NN
```

3 To have the currency symbol always print next to the leftmost digit, place the $ sign next to the zero-suppress 0 in the edit word. A currency symbol that changes positions depending upon the number of positions that are zero-suppressed is known as a floating currency symbol. The SPRICE field can look like any of the following:

```
$NNN.NN
 $NN.NN
  $N.NN
   $.NN
```

Chapter 15. Alternate Collating Sequence and File Translation

Each of the 256 EBCDIC characters is represented internally by a hexadecimal value (see Figure 15-1 on page 15-3). The order (ascending or descending sequence) in which these characters appear in the computer is based on this hexadecimal value and is known as the normal collating sequence. The alternate collating sequence function can be used to alter the normal collating sequence. This function also can be used to allow two or more characters to be considered equal.

Changing the Collating Sequence

Using an alternate collating sequence means modifying the collating sequence for character match fields (file selection) and character compares. The calculation operations affected by the alternate collating sequence are COMP, CABxx, DOUxx, DOWxx, and IFxx. The characters are not permanently changed by the alternate collating sequence, but are temporarily altered until the matching field or character compare operation is completed.

Changing the collating sequence does not affect the LOKUP and SORTA operations or the hexadecimal values assigned to the figurative constants *HIVAL and *LOVAL. However, changing the collating sequence can affect the order of the values of *HIVAL and *LOVAL in the collating sequence. Therefore, if you specify an alternate collating sequence in your program and thereby cause a change in the order of the values of *HIVAL and *LOVAL, undesirable results may occur.

Specifying an Alternate Collating Sequence

To specify that an alternate collating sequence is to be used, enter an S in position 26 in the control specification. Describe each character whose collating sequence is to be changed on the Translation Table and Alternate Collating Sequence Coding Sheet (see Figure 15-1 on page 15-3). Then transcribe the sequence changes into the correct record format for entry into the system. These records, called the alternate collating sequence table records, must be entered after the RPG/400 specifications and, if used, after the file translation table records. However, the alternate collating sequence table records must be entered before arrays and tables loaded at compile time.

Filling Out the Alternate Collating Sequence Coding Sheet

Specify each change in the collating sequence in the *Replaced By* column on the Translation Table and Alternate Collating Sequence Coding Sheet. In the *Graphic* column find the character whose normal sequence is to be changed. Enter the hexadecimal value of this character in the *Replaced By* column of the replacing character.

For example, if a blank is to be replaced by (or be considered the same as) a zero, enter F0 (the hexadecimal value of zero) in the *Replaced By* column next to the hexadecimal value of a blank (40). Therefore, whenever the system reads a blank

hexadecimal value of a blank (40). Therefore, whenever the system reads a blank and uses it in a character compare or match field, the blank is considered the same as a zero.

If a character is to be inserted between two consecutive characters, you must specify every character that is altered by this insertion. For example, if the dollar sign ($) is to be inserted between A and B, specify the changes for character B through I on the coding sheet.

The Translation Table and Alternate Collating Sequence Coding Sheet shows many hexadecimal values between the characters I and }, R and \, and Z and 0 that have no printable graphic associated with them. Because of this arrangement of non-graphics in the collating sequence, a character such as $, when inserted between A and B, changes only the position of characters B through I. Because the I replaces the nongraphic represented by hexadecimal CA, the remaining characters in the collating sequence are not affected.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

International Business Machines Corporation

G321-9096- U/M 050*
Printed in U.S.A.
*No. of sheets per pad may vary slightly

Code	Graphic	Entry	Replaced By/Takes Place Of
00000000		00	
00000001		01	
00000010		02	
00000011		03	
00000100		04	
00000101		05	
00000110		06	
00000111		07	
00001000		08	
00001001		09	
00001010		0A	
00001011		0B	
00001100		0C	
00001101		0D	
00001110		0E	
00001111		0F	
00010000		10	
00010001		11	
00010010		12	
00010011		13	
00010100		14	
00010101		15	
00010110		16	
00010111		17	
00011000		18	
00011001		19	
00011010		1A	
00011011		1B	
00011100		1C	
00011101		1D	
00011110		1E	
00011111		1F	
00100000		20	
00100001		21	
00100010		22	
00100011		23	
00100100		24	
00100101		25	
00100110		26	
00100111		27	
00101000		28	
00101001		29	
00101010		2A	
00101011		2B	
00101100		2C	
00101101		2D	
00101110		2E	
00101111		2F	
00110000		30	
00110001		31	
00110010		32	

Code	Graphic	Entry	Replaced By/Takes Place Of
00110011		33	
00110100		34	
00110101		35	
00110110		36	
00110111		37	
00111000		38	
00111001		39	
00111010		3A	
00111011		3B	
00111100		3C	
00111101		3D	
00111110		3E	
00111111	b	3F	
01000000	Blank	40	
01000001		41	
01000010		42	
01000011		43	
01000100		44	
01000101		45	
01000110		46	
01000111		47	
01001000		48	
01001001		49	
01001010	¢	4A	
01001011	.	4B	
01001100	<	4C	
01001101	(	4D	
01001110	+	4E	
01001111	|	4F	
01010000	&	50	
01010001		51	
01010010		52	
01010011		53	
01010100		54	
01010101		55	
01010110		56	
01010111		57	
01011000		58	
01011001		59	
01011010	!	5A	
01011011	$	5B	
01011100	*	5C	
01011101	)	5D	
01011110	;	5E	
01011111	¬	5F	
01100000	-	60	
01100001	/	61	
01100010		62	
01100011		63	
01100100		64	
01100101		65	

Code	Graphic	Entry	Replaced By/Takes Place Of
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010	¦	6A	
01101011	,	6B	
01101100	%	6C	
01101101	_	6D	
01101110	>	6E	
01101111	?	6F	
01110000		70	
01110001		71	
01110010		72	
01110011		73	
01110100		74	
01110101		75	
01110110		76	
01110111		77	
01111000		78	
01111001	`	79	
01111010	:	7A	
01111011	#	7B	
01111100	@	7C	
01111101	'	7D	
01111110	=	7E	
01111111	"	7F	
10000000		80	
10000001	a	81	
10000010	b	82	
10000011	c	83	
10000100	d	84	
10000101	e	85	
10000110	f	86	
10000111	g	87	
10001000	h	88	
10001001	i	89	
10001010		8A	
10001011		8B	
10001100		8C	
10001101		8D	
10001110		8E	
10001111		8F	
10010000		90	
10010001	j	91	
10010010	k	92	
10010011	l	93	
10010100	m	94	
10010101	n	95	
10010110	o	96	
10010111	p	97	
10011000	q	98	

Code	Graphic	Entry	Replaced By/Takes Place Of
10011001	r	99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001	~	A1	
10100010	s	A2	
10100011	t	A3	
10100100	u	A4	
10100101	v	A5	
10100110	w	A6	
10100111	x	A7	
10101000	y	A8	
10101001	z	A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000	{	C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	

Code	Graphic	Entry	Replaced By/Takes Place Of
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000	\	E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

Note: Not all graphic symbols shown here are available on all systems.

Figure 15-1. Translation Table and Alternate Collating Sequence Coding Sheet

Formatting the Alternate Collating Sequence Records

The changes to the collating sequence that are described on the coding sheet must be transcribed into the correct record format so that they can be entered into the system. The alternate collating sequence must be formatted as follows:

Record Position	Entry
1-6	ALTSEQ (This indicates to the system that the normal sequence is being altered.)
7-8	Leave these positions blank.
9-10	Enter the hexadecimal value for the character whose normal sequence is being changed (taken from the *Entry* column of the Translation Table and Alternate Collating Sequence Sheet).
11-12	Enter the hexadecimal value of the character replacing the character whose normal sequence is being changed (taken from the *Replaced By* column of the Translation Table and Alternate Collating Sequence Coding Sheet).
13-16 17-20 21-24 ... 77-80	All groups of four beginning with position 13 are used in the same manner as positions 9 through 12. In the first two positions of a group enter the hexadecimal value of the character to be replaced. In the last two positions enter the hexadecimal value of the character that replaces it.

The records that describe the alternate collating sequence must be preceded by a record with **b (b = blank) in positions 1 through 3. The remaining positions in this record can be used for comments.

File Translation

The file translation function translates any of the 256 8-bit codes used for EBCDIC characters into another 8-bit code (see Figure 15-1 on page 15-3). The use of file translation indicates one or both of the following:

- A character code used in the input data must be translated into AS/400 code.
- The output data must be translated from AS/400 code into a different code. The translation on input or output data occurs *after* any editing or field selection has taken place.

If file translation is used to translate data in an update file, each record must be written before the next record is read.

Remember the following when specifying file translation:

- File translation can be specified for data in array or table files (T in position 16 of the file description specifications).
- File translation can be used with data in combined or update files that are translated at input and output time according to the file translation table provided.

If file translation is used to translate data in an update file, each record must be written before the next record is read.

- For any I/O operation that specifies a search argument in factor 1 (such as CHAIN, DELET, READE, SETGT, or SETLL) for files accessed by keys, the search argument is translated before the file is accessed.
- If file translation is specified for both a record address file and the file being processed (if the file being processed is processed sequentially within limits), the records in the record address file are first translated according to the file translation specified for that file, and then the records in the file being processed are translated according to the file translation specified for that file.

Specifying a File Translation

To specify file translation, enter an F in position 43 of the control specification. Then describe the character that is to be translated on the Translation Table and Alternating Collating Sequence Coding Sheet. The translations described must also be transcribed into the correct record format for entry into the system. These records, called the file translation table records, must precede any alternate collating sequence records, or arrays and tables loaded at compile time. They must be preceded by a record with **b (b = blank) in positions 1 through 3. The remaining positions in this record can be used for comments.

Translating One File or All Files

File translation table records must be formatted as follows:

Record Position	Entry
1-8 (to translate all files)	Enter *FILESbb (b represents a blank) to indicate that all files are to be translated. Complete the file translation table record beginning with positions 9 and 10. If *FILESbb is specified, no other file translation table can be specified in the program.
1-8 (to translate a specific file)	Enter the name of the file to be translated. Complete the file translation table record beginning with positions 9 and 10. The *FILESbb entry is *not* made in positions 1 through 8 when a specific file is to be translated.
9-10	Enter the hexadecimal value of the character to be translated from on input or to be translated to on output.
11-12	Enter the hexadecimal equivalent of the internal character RPG/400 works with. It will replace the character in positions 9 and 10 on input and be replaced by the character in positions 9 and 10 on output.
13-16 17-20 21-24 ... 77-80	All groups of four beginning with position 13 are used in the same manner as positions 9 through 12. In the first two positions of a group, enter the hexadecimal value of the character to be replaced. In the last two positions, enter the hexadecimal value of the character that replaces it.

The first blank entry ends the record. There can be one or more records per file translation table. When multiple records are required in order to define the table, the same file name must be entered on all records. A change in file name is used to separate multiple translation tables. An *FILES record causes all files, including tables and arrays specified by a T in position 16 of the file description specifications, to be translated by the same table.

Translating More than One File

If the same file translation table is needed for more than one file but not for all files, two types of records must be specified. The first record type specifies the file using the tables, and the second record type specifies the table. More than one record for each of these record types can be specified. A change in file names is used to separate multiple translation tables.

Specifying the Files

File translation table records must be formatted as follows:

Record Position	Entry
1-7	*EQUATE
8	Leave this position blank.
9-80	Enter the name(s) of file(s) to be translated. If more than one file is to be translated, the file names must be separated by commas.

Additional file names are associated with the table until a file name not followed by a comma is encountered. A file name cannot be separated between two records; a comma following a file name must be on the same record as the file name. You can create only one file translation table by using *EQUATE.

Specifying the Table

File translation table records must be formatted as follows:

Record Position	Entry
1-7	*EQUATE
8	Leave this position blank.
9-10	Enter the hexadecimal value of the character to be translated from on input or to be translated to on output.
11-12	Enter the hexadecimal equivalent of the internal character RPG/400 works with. It will replace the character in positions 9 and 10 on input and be replaced by the character in positions 9 and 10 on output.

Record Position	Entry
13-16 17-20 21-24 ... 77-80	All groups of four beginning with position 13 are used the same way as positions 9 through 12. In the first two positions of a group, enter the hexadecimal value of the character to be replaced. In the last two positions, enter the hexadecimal value of the character that replaces it.

The first blank record position ends the record. If the number of entries exceeds 80 positions, duplicate positions 1 through 8 on the next record and continue as before with the translation pairs in positions 9 through 80. All table records for one file must be kept together.

The records that describe the file translation tables must be preceded by a record with **b (b = blank) in positions 1 through 3. The remaining positions in this record can be used for comments.

Chapter 16. Auto Report Feature

The RPG/400 auto report function is a program that operates before the RPG/400 compiler. Auto report on the AS/400 System is intended to be used for conversion of existing auto report programs. The use of auto report with AS/400 RPG/400 enhancements such as externally described files or the DEFN operation code may yield undesirable results.

Group Printing

In group printing, data is summarized for a group of input records and only totals are printed on the report. Totals can have subtotals with a final total or only a final total.

Specifications

To specify group printing using auto report, enter a T in position 15 and *AUTO in positions 32 through 36. A control level indicator can be specified in positions 23 through 31. When a T-*AUTO specification is used, a line is not printed for each individual record that is read, but only after a complete control group is read.

Fields and constants defined by field description specifications that have a blank or B in position 39 and follow a T-*AUTO record description are printed on the lowest level total line. Fields defined with an A in position 39 are not printed on the total lines, but the total fields created by auto report are. Generated calculations are printed on their associated total lines. Continued column headings (C in position 39) and total-indicated fields (1 through 9 or R in position 39) can also be specified by field descriptions following a T-*AUTO record description.

Output indicators can be entered in positions 23 through 31 of a field description specification following a T-*AUTO record description if position 39 of the field description specifications contains a blank or a B. If output indicators are used in a field description that has an A in position 39 following a T-*AUTO specification, those indicators are ignored by auto report. Output indicators cannot be used in a field description that contains C, 1 through 9, or R in position 39.

Examples

Figure 16-1 on page 16-2 shows the file description and input specifications for the group printed reports shown in Figure 16-2 on page 16-3 and Figure 16-3 on page 16-4. BRANCH and REGION are defined as control fields.

Figure 16-2 on page 16-3 shows the output specifications and the group printed report showing sales totals for a company. Because the T-*AUTO specification is conditioned by L2, only the totals for REGION (L2) and for the entire company (LR) are printed on the report. The totals for BRANCH (L1) are not printed.

File Description Specifications

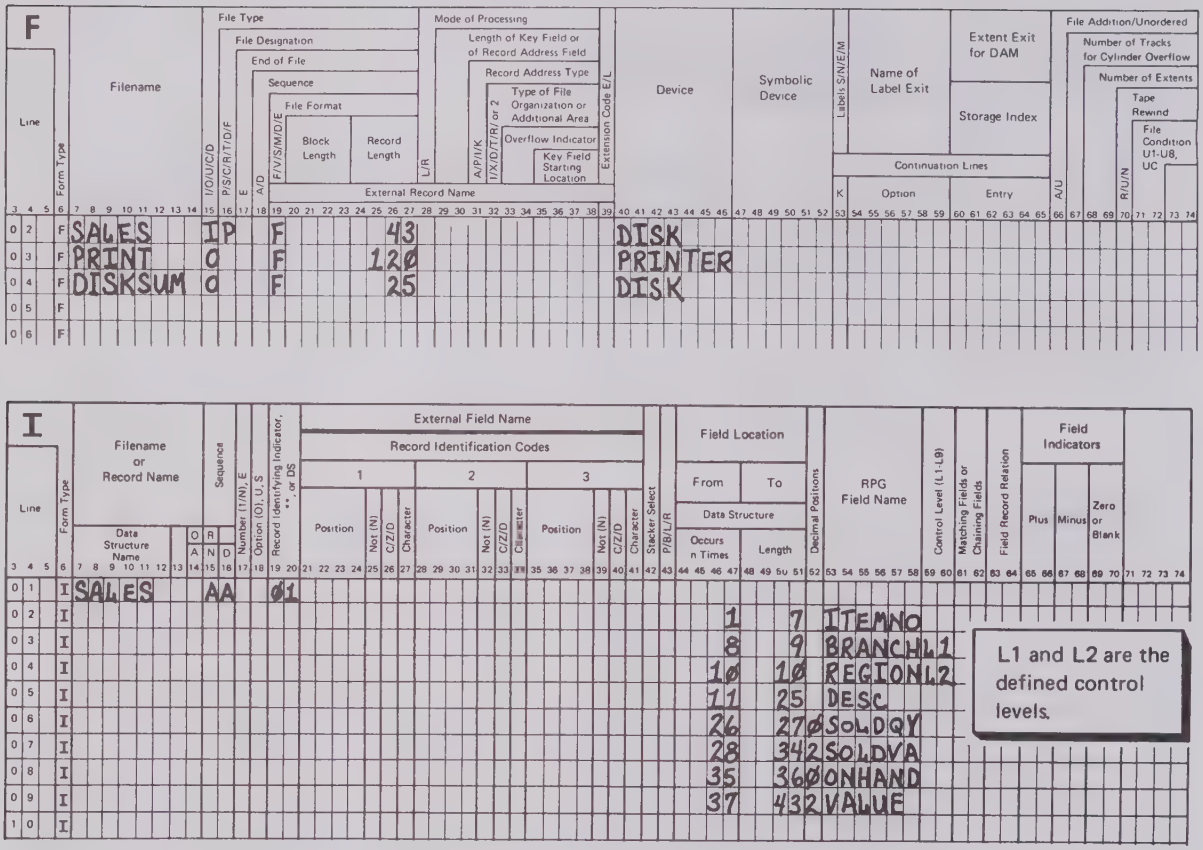
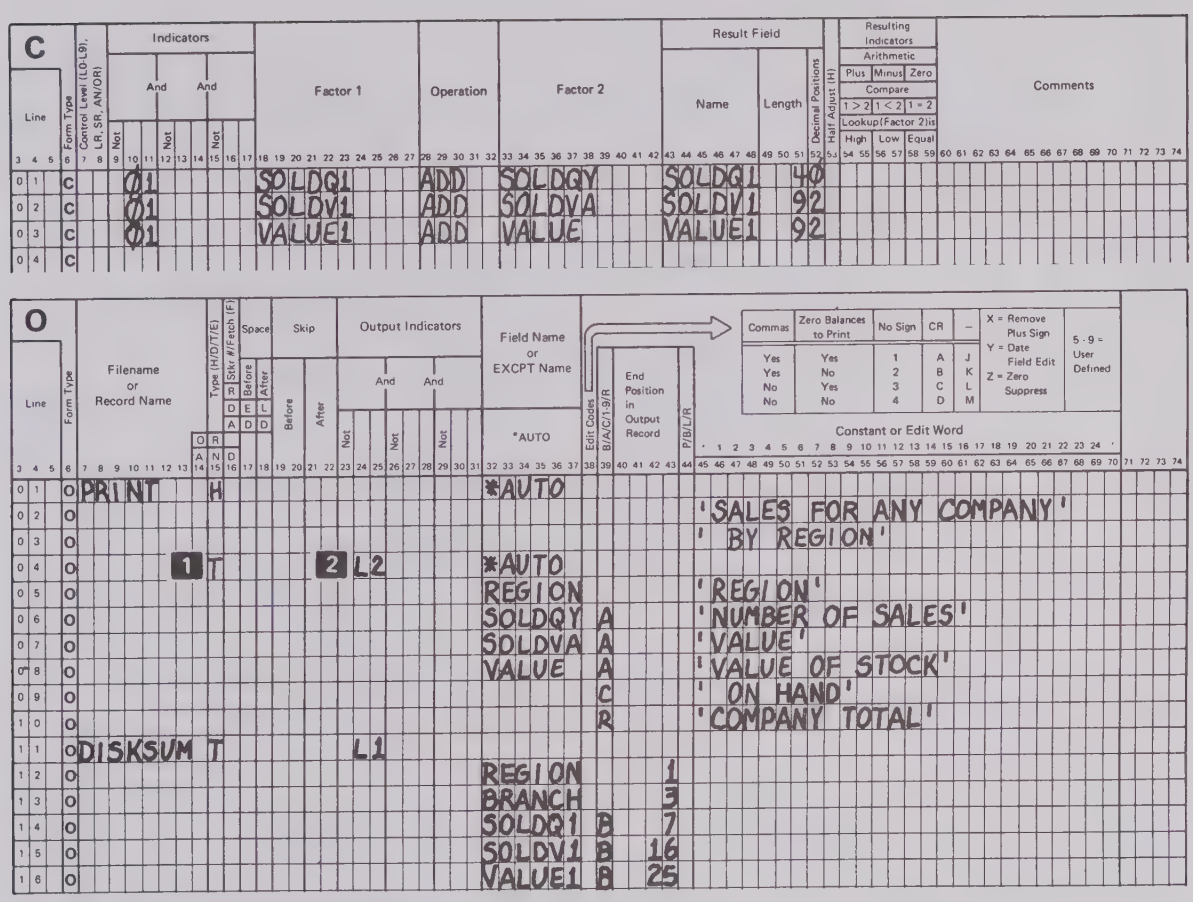


Figure 16-1. File Description and Input Specifications for Group Printed Reports

A summary file, DISKSUM, is also produced by this program. The summary file contains a summary record of the sales data for each branch. The output specifications for DISKSUM illustrate the use of standard RPG/400 output specifications in the same program with *AUTO specifications. The output record described is written on the file, DISKSUM, when there is an L1 control break (BRANCH field changes). Because the T-*AUTO specification is conditioned by L2, auto report does not generate fields for the L1 control level. Therefore, standard RPG/400 calculation specifications must be used to calculate the L1 totals. The L1 total fields that are written on the DISKSUM file (SOLDQ1, SOLDV1, and VALUE1) must be defined in the calculations.

Figure 16-3 on page 16-4 shows a group printed report similar to the one shown in Figure 16-2 on page 16-3. However, the T-*AUTO specifications are not conditioned by a control level indicator, so totals are printed for all defined control levels and for LR.



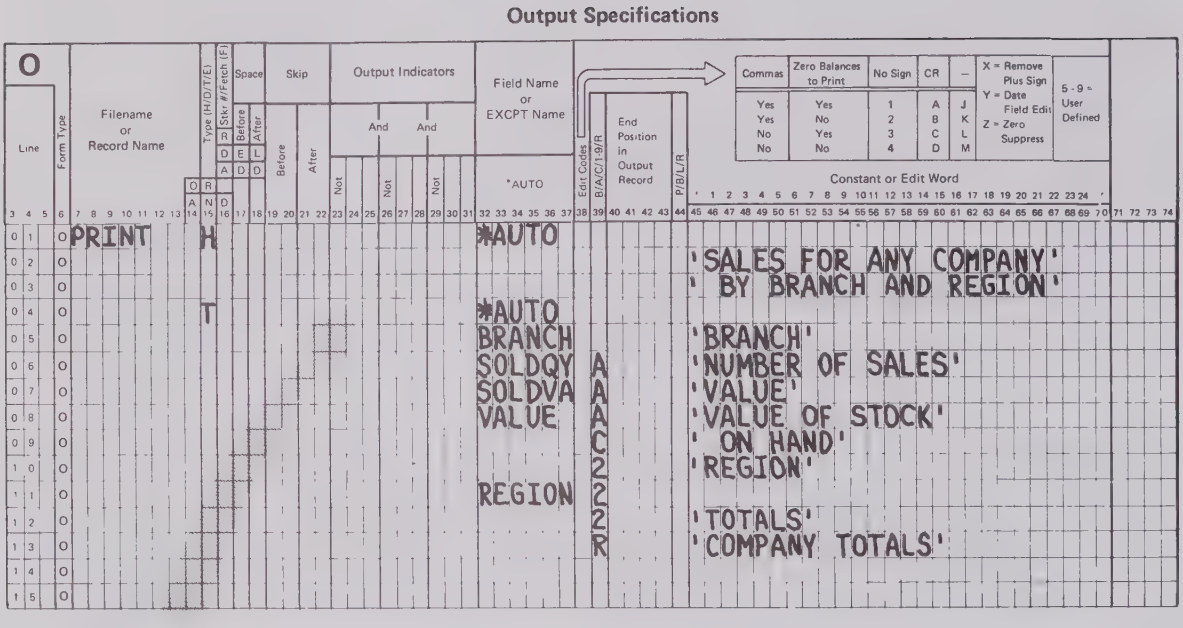
1 T in position 15 with *AUTO in positions 32 through 37 specifies a group printed report.

2 Because L2 is entered under output indicators, total lines are printed only for L2 and LR, although L1 is also a defined control level.

3 In group printing, the lowest level total lines printed (L2, in this case) are single-spaced, like detail lines.

```
11/11/87        SALES FOR ANY COMPANY BY REGION          PAGE        1

      REGION    NUMBER OF SALES        VALUE        VALUE OF STOCK
                                                        ON HAND

           1                 23        71,000.00        19,000.00    *
           3                 30        70,000.00        29,000.00    *

COMPANY TOTAL                53       141,000.00        48,000.00   **
```

Figure 16-2. Using *AUTO to Produce a Group Printed Report Showing Region and Final Totals



When no control level indicators are entered under output indicators, a total line is generated for each defined control level indicator (L1 and L2, in this case) and for LR.

```
                SALES FOR ANY COMPANY BY BRANCH AND REGION      PAGE    1

                BRANCH    NUMBER OF SALES      VALUE       VALUE OF STOCK
                                                               ON HAND
L1 ->            17              17          53,000.00        12,000.00    *
L2 ->            22               6          18,000.00         7,000.00    *
REGION 1 TOTALS                  23          71,000.00        19,000.00    **
L1 ->            25              30          70,000.00        29,000.00    *
REGION 3 TOTALS                  30          70,000.00        29,000.00    **
COMPANY TOTALS                   55         141,000.00        48,000.00    ***
LR ->
```

Figure 16-3. Using *AUTO to Produce a Group Printed Report Showing Region, Branch, and Final Totals

/COPY Statement Specifications

The auto report copy function provides a way to include RPG/400 source specifications from a source file member in an RPG/400 program. Use the copy function to include source specifications that are identical or nearly identical in several different programs, thereby reducing the need to repeatedly code specifications that are used in several programs. For example, if file description and input specifications for a

particular file are similar in different programs, these specifications can be placed in a source file member and included in any program by the copy function.

Auto report specifications and any valid RPG/400 specifications, including arrays and tables can be copied in this manner. The auto report option specifications and other copy statements cannot be copied. See "Examples of Using Auto Report" on page 16-29 for an example of using the copy function.

The specifications included in an auto report program by the copy function are initially placed in the program immediately following the /COPY statement. When all specifications are copied from the source file member, the entire auto report program is sorted into the order required by the RPG/400 compiler.

To request the copy function, use the /COPY statement. This statement identifies the source file member containing the RPG/400 specifications to be included in the source program generated by auto report. /COPY statements must follow the auto report option specifications, and they must precede source arrays and tables (file translation tables, alternate collating sequence tables, and compile-time arrays and tables).

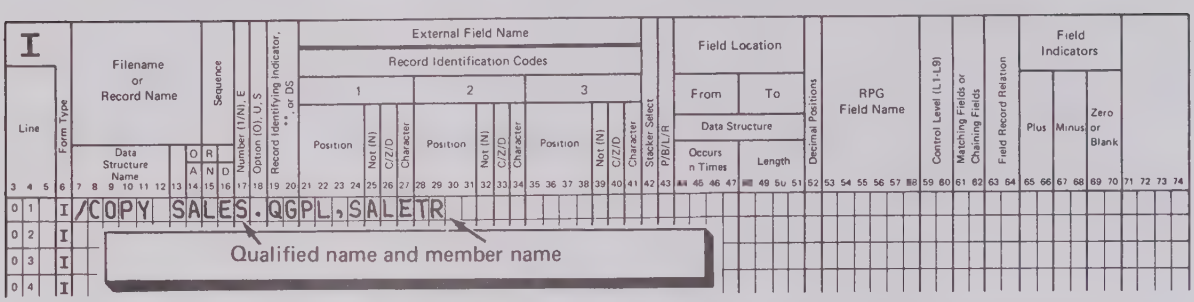
The file name specified on a /COPY statement must not be changed by a control language override command. No inline data file can be specified as the file on a /COPY statement.

The auto report /COPY specification is similar in syntax to the compiler /COPY directive.

The format of the /COPY statement is:

Position	Entry
1-5	Page and line number indicating the placement of the statement in the sequence of auto report source specifications.
6	This position can contain any entry except H or U, or can be blank.
7-11	Enter the characters /COPY.
12	Blank.
13-44	Enter the qualified file name (library-name/file-name in the AS/400 Native Environment, or file-name.library-name in the System/38 Environment), followed by a comma, followed by the member name. If the library name is not specified, the library list (*LIBL) is used to locate the file. If F1, F2, R1, or R2 is specified as the file name, the file name QRPGSRC is assumed and the library list is used to locate the file. If only one entry appears, it is the member name; the file name QRPGSRC is assumed and the library list is used to locate the file. The member name must exist in QRPGSRC.
45-49	Blank.
50-80	Enter any information or comments. The contents of these positions are not read by auto report.

Figure 16-4 on page 16-6 shows an example of the /COPY statement.



Note: It is convenient to code the /COPY statement on the input specifications if input specifications are to be modified as they are copied.

Figure 16-4. Example of the /COPY Auto Report Statement

Modifying Copied Specifications

Statements can be included in the auto report specifications to modify file description and input field specifications as they are copied from the source file member. No other types of specification can be modified. /COPY modifier statements from the source program that add, change, or delete entries on input field specifications are identified by an X in print position 6 of the auto report listing.

Modifying File Description Specifications

To modify a file description specification that is copied from a source file member, enter the file name in positions 7 through 14 of a file description specification (F in position 6). Then make only those entries on the line that are to replace existing entries in the copied specification or that are to be included as new entries. Blank entries in the modifier statement do not affect the copied statement.

For example, the file description specifications for a frequently used file named SALES is to be copied from a source file member. The original specification contains an I in file type (position 15), defining SALES as an input file (see Figure 16-5 on page 16-8). To update the sales file, change position 15 to a U by including a modifier file description specification in the auto report source program. The modifier statement must contain the file name, SALES, and the new file type entry, U. As a result of the modifier statement, the file type on the copied file description specification is changed from I to U.

To set an entry to blanks, enter an ampersand (&) in the first position of that entry of the modifier statement, and leave the remaining positions blank. For example, to remove the overflow indicator (positions 33 and 34) from the specification shown in Figure 16-5 on page 16-8, add an ampersand to the modifier statement in position 33, as shown in Figure 16-6 on page 16-8, and leave position 34 blank.

Modifier statements for file description specifications do not have to be in any particular order in the auto report source program, except that they cannot immediately follow the /COPY statement if input field specifications are also being modified.

No modifications are allowed to the file description continuation specifications that accompany a copied file description. To add new continuation specifications, place them after a file description modifier statement for the file. A maximum of five continuation specifications are allowed to follow a file description specification (combined total of original and added continuation specifications).

Modifying Input Field Specifications

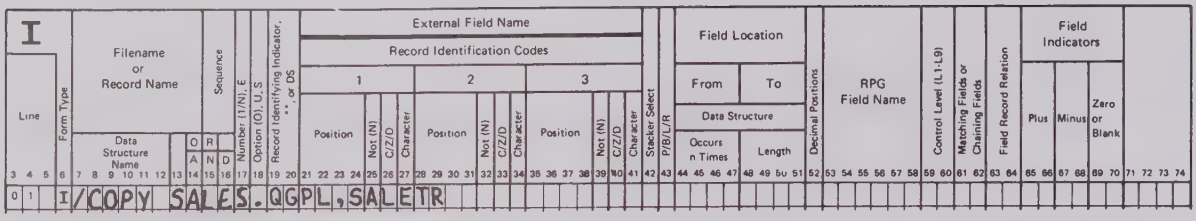
Only input field specifications (specifications describing individual fields on the input record) can be modified. To modify a copied input field specification, enter the field name in positions 53 through 58 of an input field modifier statement (I in position 6). Modifier statements for input field specifications must immediately follow the /COPY statement in the auto report program that copies those specifications. The first specification following the /COPY statement that is not an input field specification is considered the end of the input field modifier statements for the /COPY statement. (A comment statement with an I in position 6 is not considered the end of the input field modifier statements.)

The fields that can be modified are:

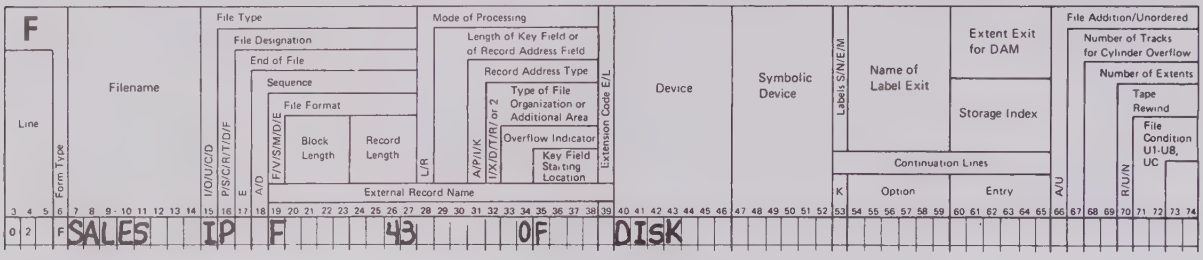
- Position 43 (packed/binary)
- Positions 44-51 (field location)
- Position 52 (decimal positions)
- Positions 59-60 (control levels)
- Positions 61-62 (matching fields)
- Positions 63-64 (field record relation)
- Positions 65-70 (field indicators).

The method of replacing, adding, or blanking entries is similar to the method used to modify file description specifications. To replace or add entries, code the new entry in the proper location in the modifier statement; to set an entry to blank, place an ampersand (&) in the first position of that entry in the modifier statement. Figure 16-7 on page 16-10 shows examples of modifying input specifications.

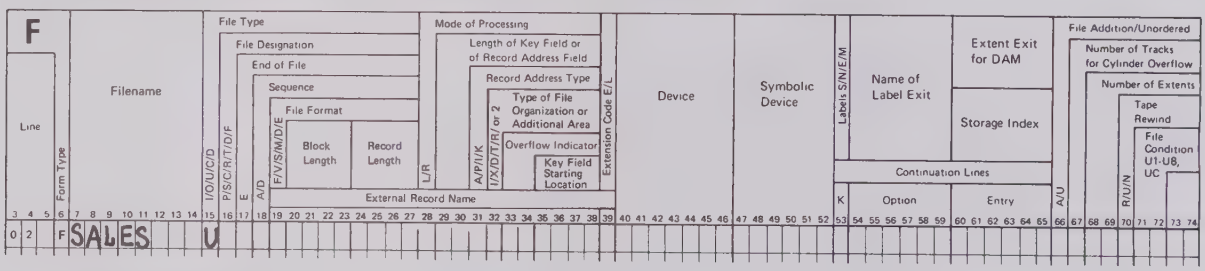
/COPY Statement Specifications



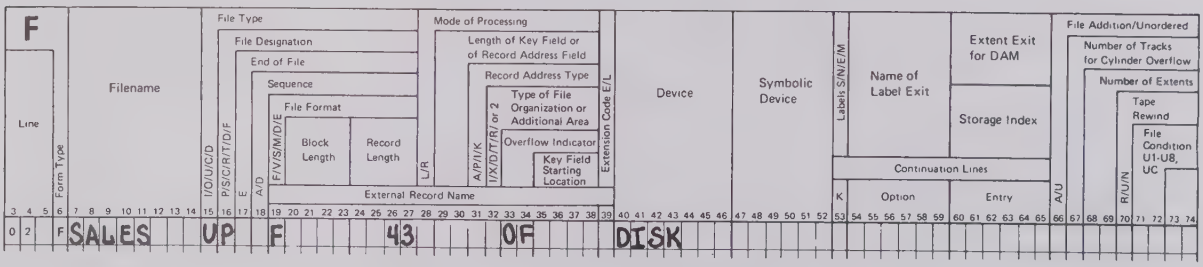
/COPY statement to copy specifications for SALES file from the library QGPL. The member name is SALETR.



File description specification as it is stored in the source file member.



Copy function modifier statement.



Resulting file description specification that is included in the RPG/400 source program.

Figure 16-5. Modifying a Copied File Description Specification

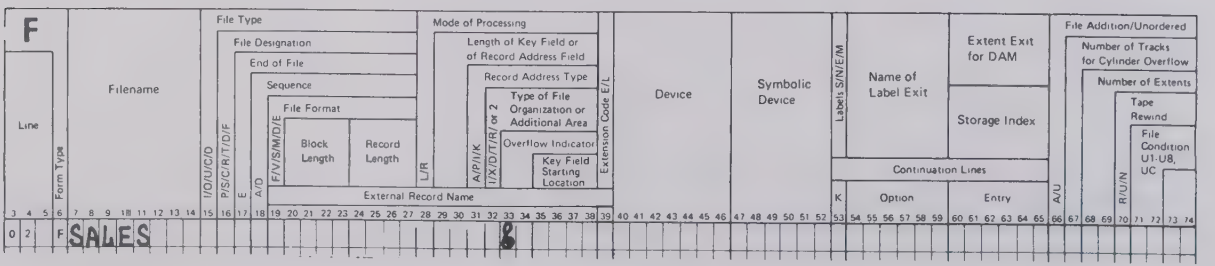
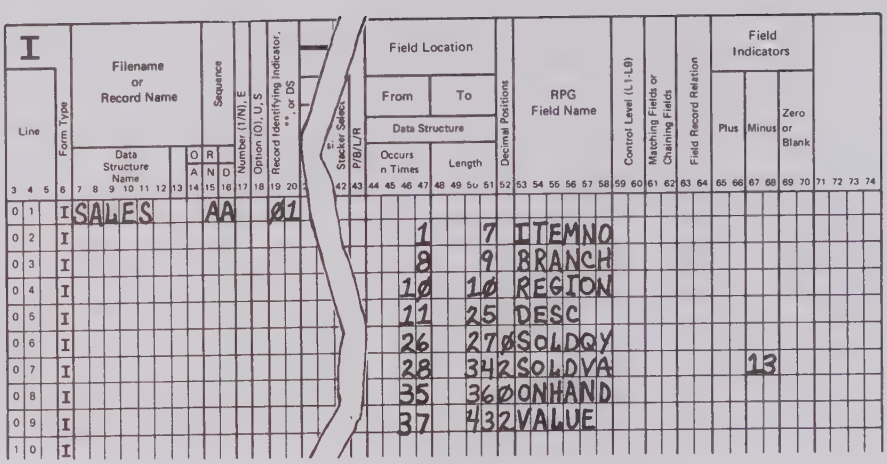


Figure 16-6. Setting a Copied File Description Entry to Blank

The modifier statement modifies all copied input field specifications that have the same field name. If there is no input field by the same name, the modifier statement is added to the program as a new input field specification. Modifier statements with duplicate field names are allowed (length and number of decimal positions must also be the same), but only the first is used to modify a copied specification. Other field names are added as new input field specifications. Up to 20 input field modifier statements are allowed per /COPY statement.

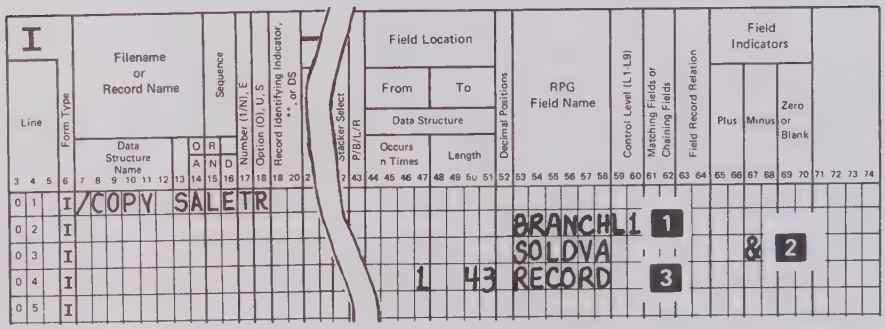
For best results, first place those statements that modify existing input field specifications; then place those that are to be added as new input field specifications. This procedure is suggested because input field modifier statements that do not fit into the special main storage table for modifier statements are added to the RPG/400 source program as new input field specifications. This order of specifying modifier statements increases the likelihood that excess statements, if any, will be valid field descriptions.

Input specifications as stored in a source file.



/COPY statement and modifier statements:

1. Add an entry to BRANCH field description
2. Blank out minus field indicator on SOLDVA description
3. Add a new field description



Resulting input specifications for SALES file showing:

1. Added L1 indicator
2. Blanks in place of minus field indicator
3. Added field description

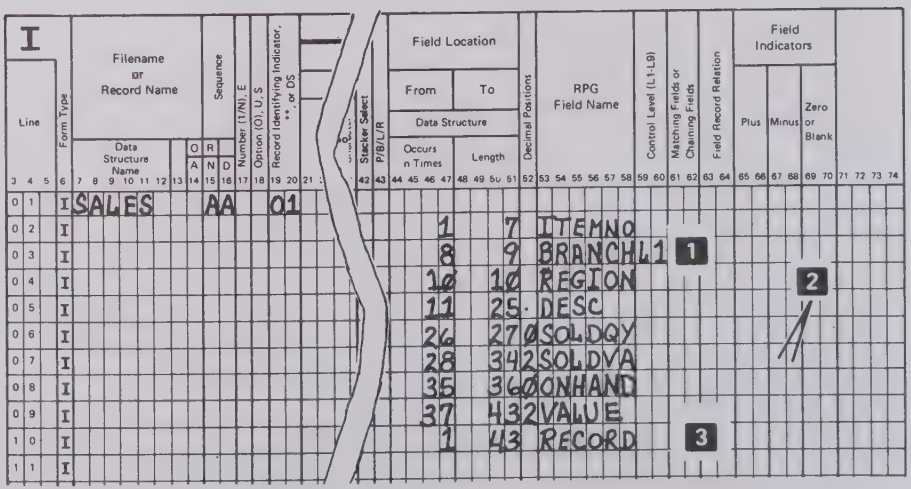


Figure 16-7. Modifying Copied Input Field Specifications

Report Format

One of the advantages of auto report is that it frees the programmer from the task of specifying the format of his report on the output specifications form. Auto report can completely format the report by spacing, skipping, centering lines, and calculating end positions for fields and constants.

Spacing and Skipping

Spacing and skipping can be either left to auto report or specified by the programmer. Figure 16-8 on page 16-13 shows spacing and skipping generated by auto report. For the specifications used to produce the report, see "Generated Specifications" on page 16-16. If positions 17 through 22 are blank on an H-*AUTO specification, a skip to line 06 is done before the first heading line is printed and space-two-after is done for the last heading line. If more than one heading line is specified, space-one-after is done for the first and all succeeding lines except the last. To specify spacing and skipping, follow the standard RPG/400 rules for spacing and skipping.

Column heading lines are spaced like page headings. Space-one-after is done for all except the last. Space-two-after is done for a single heading line, or for the last heading line if more than one is specified. Spacing and skipping entries cannot be specified for column headings. If spacing and skipping entries are made on a D-*AUTO record description specification, the entries apply to the detail line generated. The entries do not apply to column headings or total lines generated by auto report from the D-*AUTO specification. Standard RPG/400 rules for spacing and skipping must be followed. Space-one-after is assumed for the generated detail line if spacing and skipping entries are not made.

Space-two-after is generated for all total lines produced by auto report from a D-*AUTO specification. In addition, the lowest level total line and the final total line are also generated with a space-one-before.

If spacing and skipping entries are made on a T-*AUTO specification, the entries apply to the lowest level total line generated, but not to column headings or higher level total lines. If spacing and skipping are not made, the lowest level total lines are generated with space-one-after; all higher levels are generated with space-two-after. Space-one-before is always generated for the second-to-the-lowest level total and the final total (see Figure 16-3 on page 16-4 for an example).

Placement of Headings and Fields

Auto report generates end positions for fields and constants and centers column headings, columns, and report lines (see Figure 16-8 on page 16-13 for an example). However, if an end position is specified for a field or constant on a D/T-*AUTO field description line, that end position is used on all column heading, detail, and total specifications generated from the field description. (The specified end position may be altered slightly by auto report when the line is centered or when the column heading and field are positioned in relation to each other.) If the specified end position causes an overlay with a previous field or constant, auto report generates a new end position.

Specify end positions only to eliminate the automatic spacing between fields or to spread out or expand a report on the page.

Page Headings

If the date and page number are printed on the first *AUTO page heading line (that is, if they are not suppressed by an N in position 27 of the option specifications or by the *NODATE option of the RPTOPT parameter in the CRTRPTPGM command), the date is always printed in positions 1 through 8. The page number is printed with an end position equal to the highest end position of the longest line in the report. When the first *AUTO page heading (including date, title, and page number) is the longest line in the report, one blank space separates the title from the date and the word PAGE from the title. If the resulting line exceeds the record length of the printer file, the excess information on the right of the line is not printed.

If a line generated from a D/T-*AUTO specification is the longest report line, that line is printed starting in print position 1, and the title portion of the first page heading line is centered in relation to that line.

Additional *AUTO page headings are then centered on the first *AUTO page heading line.

If an *AUTO page heading is the longest line in the report and a D/T-*AUTO specification is present, any other *AUTO page heading lines and the line generated from the D/T-*AUTO specification are centered on the longest page heading.

Fields and constants appear in the order specified in the *AUTO output specifications from left to right. Auto report provides one blank space before and after fields on the heading line. No spacing is provided between constants.

Reformatting *AUTO Page Headings

You can reformat an *AUTO page heading line if you do not want to use the end positions for fields and constants that are generated by auto report. If you want to find what end positions are generated for page, date, and title information, see the listing of the generated source program that is produced by the RPG/400 compiler. See "Generated Specifications" on page 16-16.

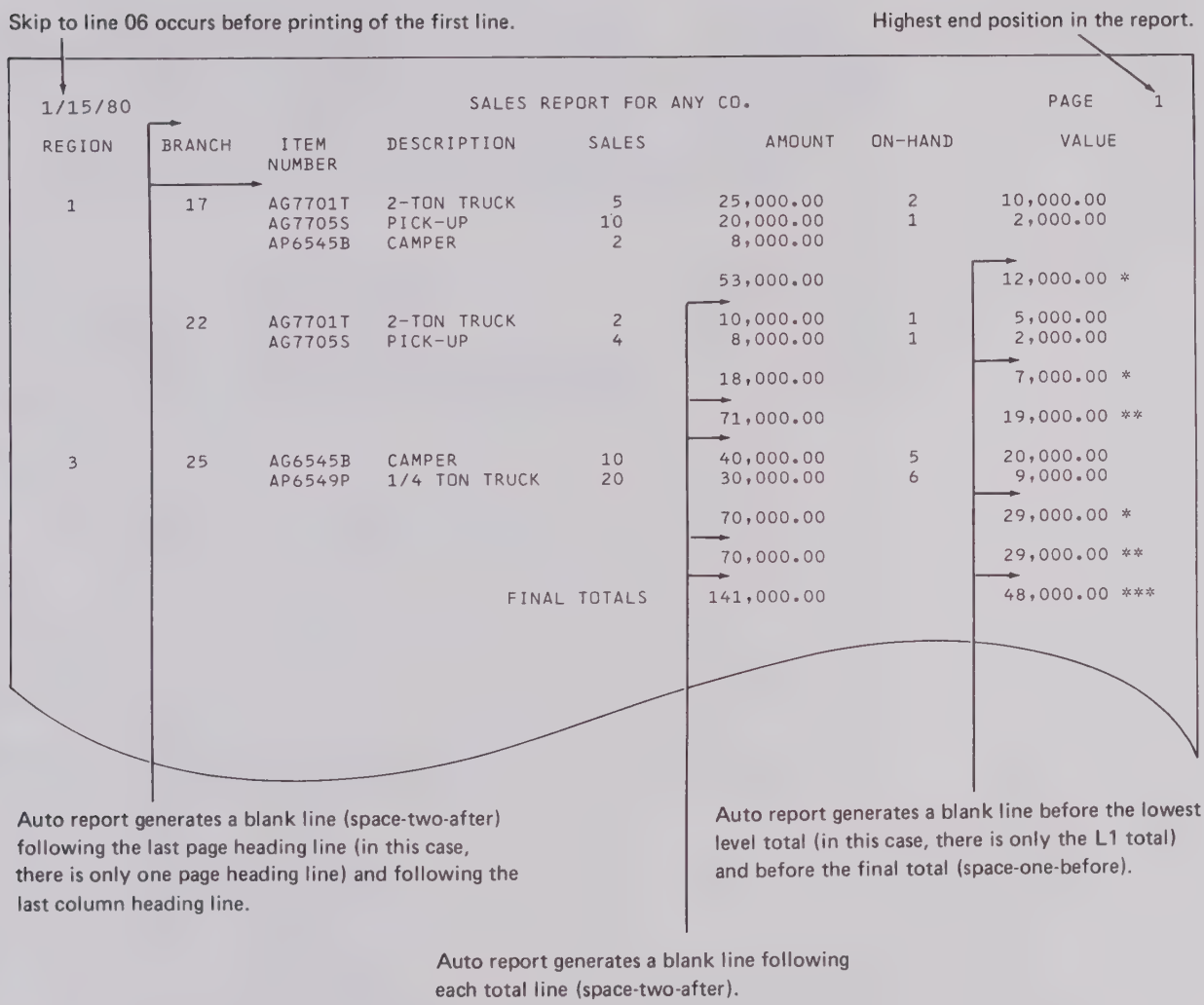


Figure 16-8. Report Illustrating Format Generated by Auto Report

Body of the Report

Placement of column headings above columns depends on which is longer, the heading or the associated field (including edit characters). If any column heading is longer than the associated field, the field is centered under the longest column heading constant. If, however, the field is longer than the longest column heading constant, the column heading is left-adjusted over a character field and right-adjusted over a numeric field. When more than one column heading line is specified, shorter column headings are always centered on the longest column heading.

Fields and constants appear from left to right on a line in the order they are specified by the output specifications. At least two blank spaces appear before each field on the line. However, no spaces are provided before a constant; the programmer must incorporate blanks within constants to provide for additional spacing.

Total indication information (fields and constants specified with 1 through 9 or R in position 39) is placed to the left of the first total field (A in position 39) on the corresponding total line, followed by two spaces. If two or more such fields or constants are specified for a total line, they appear from left to right in the order specified on the left of the first total on the line. Each field is preceded and followed by one space. No spacing is provided for constants.

Overflow of the D/T-*AUTO Print Lines

If the lines generated from a D/T-*AUTO specification are longer than the record length specified for the printer file, a second print line (overflow line) is generated for each column heading line, detail (or group print) line, and total line. (Remember, a second print line is not generated for *AUTO page heading lines.) The excess information is placed on the overflow line in the order specified, right-adjusted.

Figure 16-9 on page 16-15 shows the result of an overflow condition.

In the output specifications for the report shown in Figure 16-9 on page 16-15, no spacing or skipping is specified. If spacing and skipping were specified, however, auto report spaces the report as follows:

- Column heading lines and total lines are spaced as shown in Figure 16-9 on page 16-15.
- The space-before and skip-before entries specified are for the original detail (or group print) line. Auto report generates space-one-after for this line.
- The space-after and skip-after entries specified are for the overflow line. Auto report generates blanks for space-before and skip-before for the overflow line.

Auto report prints those columns that cannot be completely contained on the original line on overflow lines.

8/15/80 CASH RECEIPTS REGISTER PAGE 1

REGION	ACCOUNT NUMBER	ACCOUNT NAME	INVOICE NUMBER	INVOICE DATE	DATE PAID / AMOUNT PAID	AMOUNT OWED / BALANCE DUE	DISCOUNT TAKEN / EXCESS DISCOUNT	
1	11243	JONES HARDWARE	27541	7/11/80	7/21/80 23.28	23.75	.47	
1	11352	NU-STYLE CLOTHIERS	27987	7/14/80	7/26/80 40.00	87.07 47.07		
1	11886	MIDI FASHIONS INC	15771	7/04/80	7/14/80 105.08	107.22	2.14	
1	12874	ULOOK INTERIORS	25622	7/09/80	7/23/80 67.95	67.95		
1	18274	STREAMLINE PAPER INC	29703	7/21/80	7/30/80 170.55	274.03 101.10	2.38	
					REGION TOTALS 406.86	560.02 148.17	4.99	*
2	23347	RITE-BEST PENS CO	20842	7/18/80	7/20/80 10.00	15.80 5.80		
2	25521	IMPORTS OF NM	29273	7/20/80	7/27/80 585.47	797.40 200.00	11.93	
2	26723	ALRIGHT CLEANERS	19473	7/07/80	7/23/80 462.00	462.00		
2	28622	NORTH CENTRAL SUPPLY	17816	7/05/80	7/22/80 75.97	75.97		
2	29871	FERGUSON DEALERS	27229	7/10/80	7/22/80 61.91	61.91		
					REGION TOTALS 1,195.35	1,413.08 205.80	11.93	*
3	30755	FASTWAY AIRLINES	26158	7/06/80	7/19/80 725.87	742.72	16.85 1.90	
3	31275	ENVIRONMENT CONCERNS	20451	7/06/80	7/30/80 15.00	29.43 14.43		
3	32457	B SOLE SILOS	27425	7/10/80	7/20/80 110.05	110.05		
3	37945	HOFFTA BREAKS INC.	18276	7/06/80	7/23/80 47.23	47.23		
					REGION TOTALS 898.15	929.43 14.43	16.85 1.90	*
4	42622	EASTLAKE GRAVEL CO	16429	7/05/80	7/23/80 29.37	29.37		
					REGION TOTALS 29.37	29.37		*
					COMPANY TOTALS 2,529.73	2,931.90 368.40	33.77 1.90	**

Figure 16-9. Report Illustrating Overflow of D-*AUTO Print Lines

Generated Specifications

Standard RPG/400 specifications are generated by auto report and are combined with RPG/400 specifications included in the input to auto report and specifications copied from the source file member to produce the final RPG/400 source program. This section describes the generated RPG/400 specifications and the order of those specifications in the RPG/400 source program.

Figure 16-10 on page 16-18 and Figure 16-11 on page 16-19 show auto report specifications for a sales report and the resulting RPG/400 source specifications that are generated for the report. Numbers are inserted in the figures to identify the auto report functions and to show the specifications that are generated by each function.

Generated Calculations

Calculations are generated to accumulate totals for fields named on *AUTO field description specifications that have an A in position 39 (see Figure 16-12 on page 16-20).

An RPG/400 subroutine is generated to accumulate the values from these fields into the lowest-level generated total fields. The name of the subroutine is always A$$SUM. The subroutine specifications are conditioned differently, depending on whether detail or group printing is specified:

- If detail printing is specified, as in Figure 16-12 on page 16-20, the EXSR statement is conditioned by the same indicator(s) that conditions the D-*AUTO specification (01 in this example). Each ADD statement in the subroutine is conditioned by the field indicator(s) specified with the field in its field description specification (none in this example).
- If group printing is specified, the EXSR statement and all ADD statements in the subroutine are unconditioned.

Total calculations are generated to roll the total from the lowest-level defined total field through the higher-level defined total fields and the final total. The total calculation to add the total from one level to that of the next higher level is conditioned by the control level indicator corresponding to the field name of the lower level. As shown in Figure 16-12 on page 16-20, total calculations to accumulate L2 and LR totals are followed by the subroutine to accumulate the lowest level total, L1.

Generated total fields are defined (given length and number of decimal positions) when the total field is the result field in a generated calculation. In the input specifications, SOLDVA and VALUE are numeric fields defined with a length of seven and two decimal positions. Figure 16-12 on page 16-20 shows that the total fields generated from SOLDVA and VALUE are defined as two positions longer than the original fields, with the same number of decimal positions.

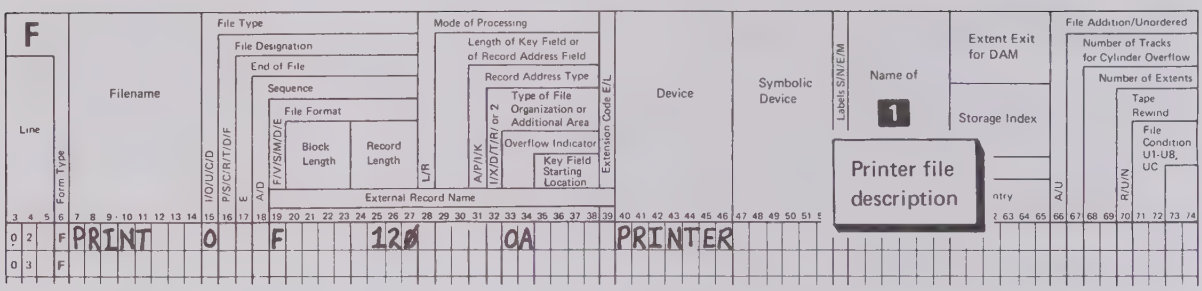
When group printing is specified (T-*AUTO specification), auto report generates total calculations to reset each of the accumulated fields (A in position 39) on the lowest level total line to zero on each cycle. A Z-ADD calculation, conditioned by L0, is generated for each accumulated field. These calculations are the first total calculations in the generated RPG/400 source program.

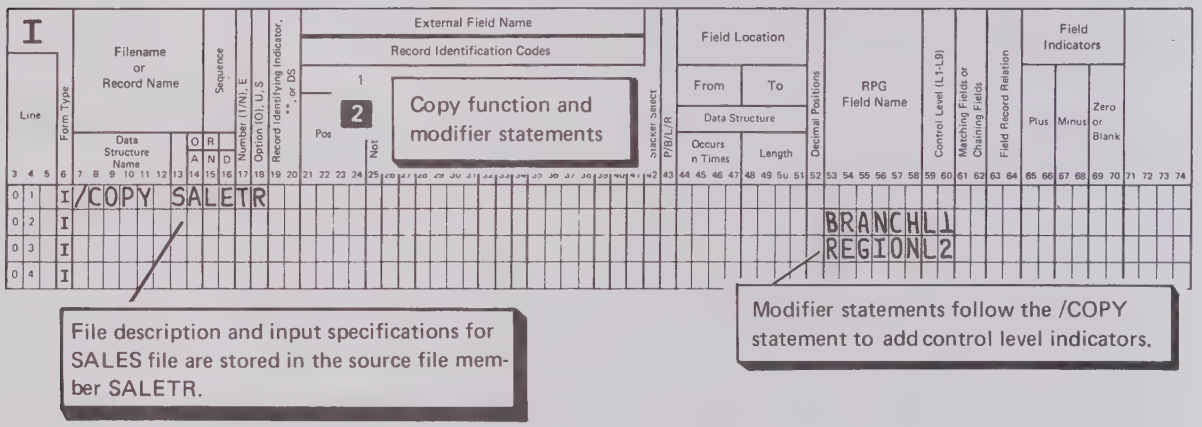
Generated Output Specifications

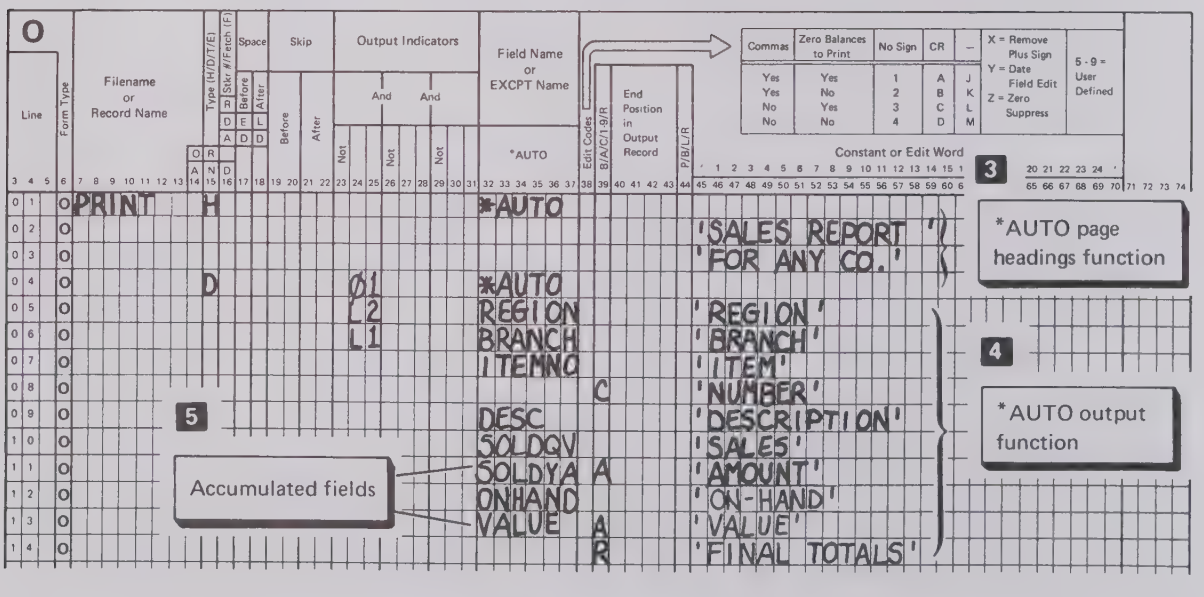
Figure 16-13 on page 16-21 shows the output specifications generated by auto report. To identify specifications supplied by auto report (column heading specifications, total specifications, conditioning indicators, spacing and skipping values, end position values, blank after) compare the listing with the auto report specifications.

Auto report generates specifications to reset accumulated fields to zero after they are printed. In this example, blank after is generated for accumulated fields.

File Description Specifications



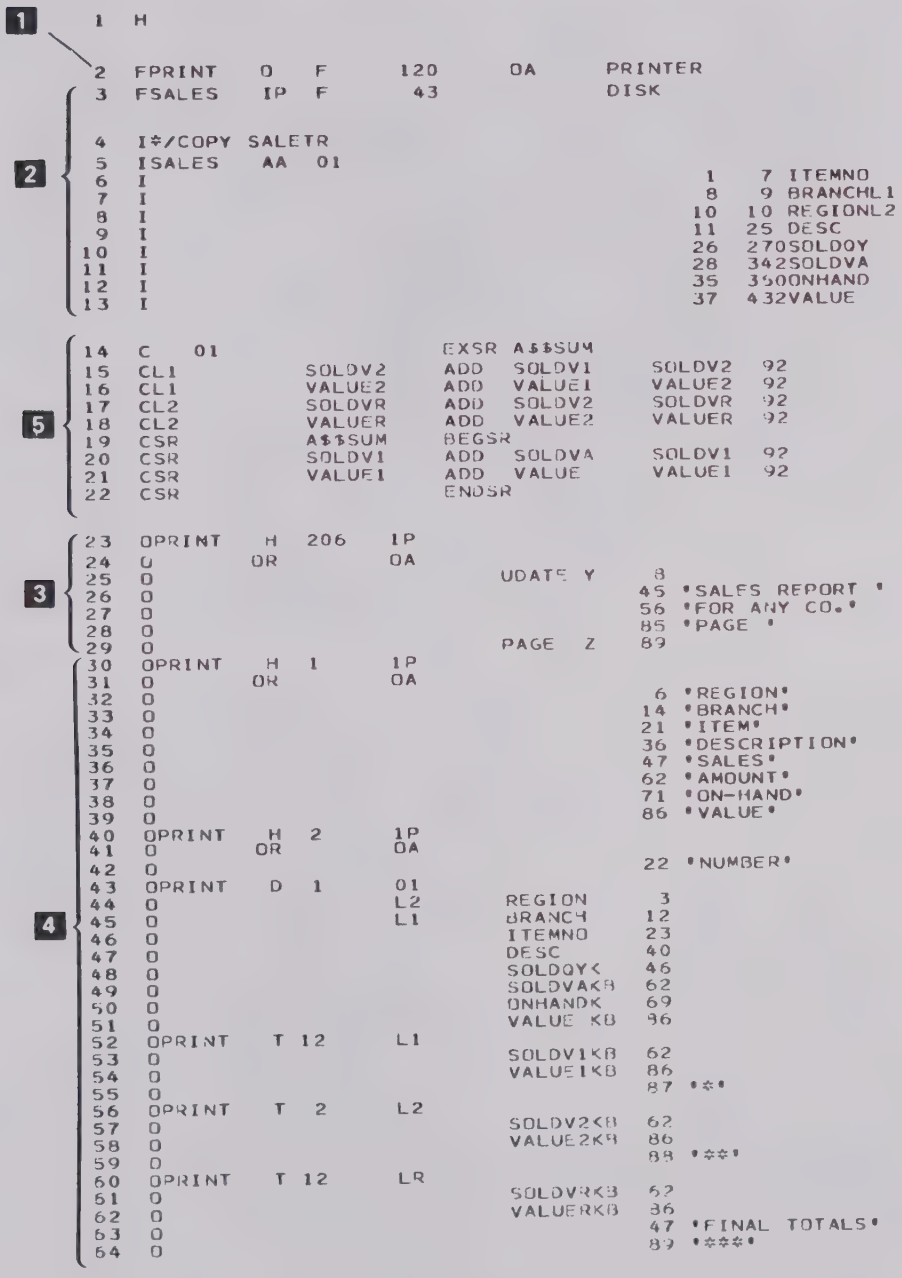




Note: The numbers refer to the corresponding numbers in the generated source program shown in Figure 16-11 on page 16-19.

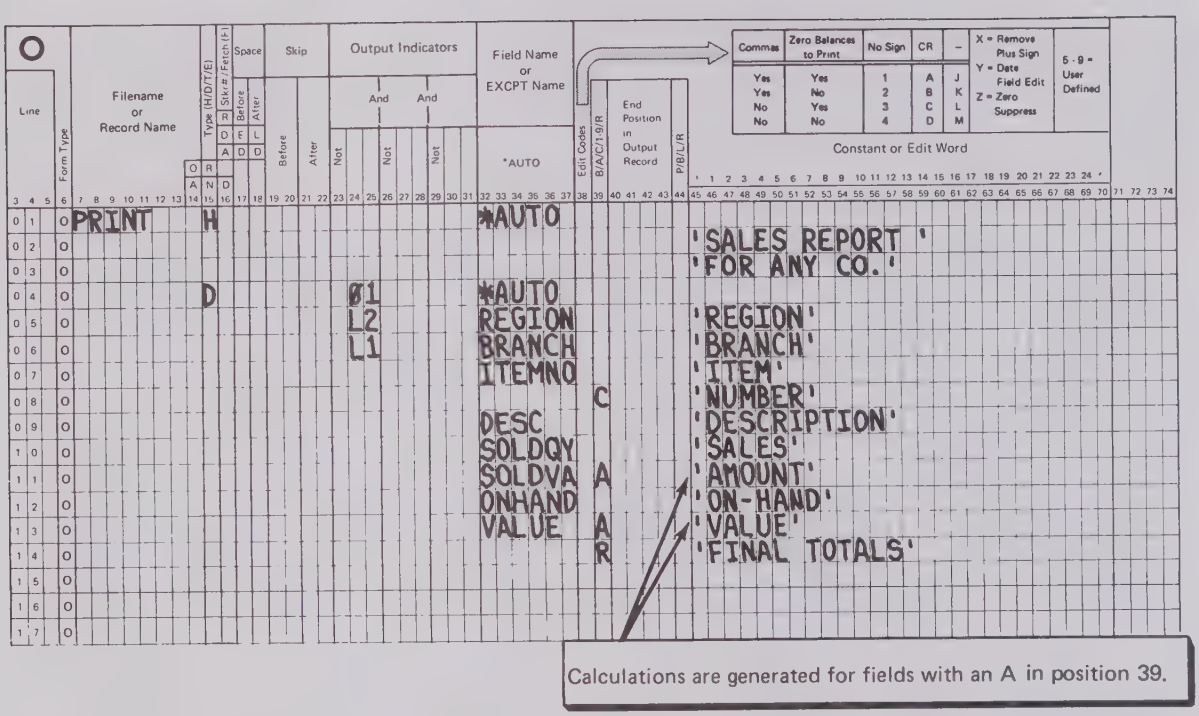
Figure 16-10. Auto Report Specifications for a Sales Transaction Report

If you do not specify a control specification, auto report generates a blank one for you.



Note: These numbers refer to the corresponding numbers shown on the auto report specifications shown in Figure 16-10.

Figure 16-11. RPG/400 Source Program Generated from Auto Report Specifications



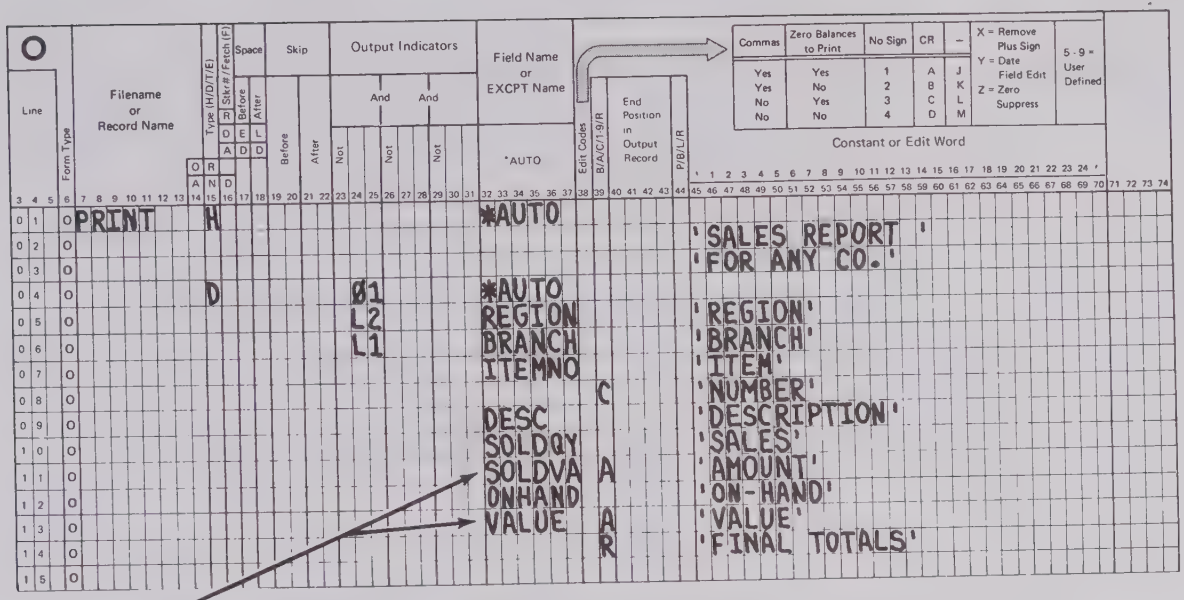
Total calculations roll higher level totals.

```
14   C     01                    EXSR A$$SUM
15   CL1         SOLDV2    ADD   SOLDV1     SOLDV2   92
16   CL1         VALUE2    ADD   VALUE1     VALUE2   92
17   CL2         SOLDVR    ADD   SOLDV2     SOLDVR   92
18   CL2         VALUER    ADD   VALUE2     VALUER   92
19   CSR         A$$SUM    BEGSR
20   CSR         SOLDV1    ADD   SOLDVA     SOLDV1   92
21   CSR         VALUE1    ADD   VALUE      VALUE1   92
22   CSR                   ENDSR
```

Subroutine accumulates the lowest level totals (L1, in this example).

Length and decimal position of generated total fields.

Figure 16-12. Calculations Generated from Auto Report Coding for Sales Transaction Report




```
23  OPRINT   H   206    1P
24  O        OR          OA
25  O
26  O
27  O
28  O
29  O
30  OPRINT   H   1      1P
31  O        OR          OA
32  O
33  O
34  O
35  O
36  O
37  O
38  O
39  O
40  OPRINT   H   2      1P
41  O        OR          OA
42  O
43  OPRINT   D   1      01
44  O                    L2
45  O                    L1
46  O
47  O
48  O
49  O
50  O
51  O
52  OPRINT   T  12      L1
53  O
54  O
55  O
56  OPRINT   T   2      L2
57  O
58  O
59  O
60  OPRINT   T  12      LR
61  O
62  O
63  O
64  O
```

Auto report generates total specifications to print accumulated totals for SOLDVA and VALUE fields.

```
UDATE Y     8
           45 'SALES REPORT '
           56 'FOR ANY CO.'
           85 'PAGE '
PAGE  Z    89

            6 'REGION'
           14 'BRANCH'
           21 'ITEM'
           36 'DESCRIPTION'
           47 'SALES'
           62 'AMOUNT'
           71 'ON-HAND'
           86 'VALUE'

           22 'NUMBER'

REGION      3
BRANCH     12
ITEMNO     23
DESC       40
SOLDQYK    46
SOLDVAKB   62
ONHANDK    69
VALUE KB   86

SOLDV1KB   62
VALUE1KB   86
           87 '*'

SOLDV2KB   62
VALUE2KB   86
           88 '**'

SOLDVRKB   62
VALUERKB   86
           47 'FINAL TOTALS'
           89 '***'
```

Two heading specifications are generated for column headings because ITEM NUMBER is a two-line heading.

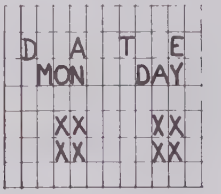
Figure 16-13. Output Specifications Generated from Auto Report Coding for Sales Transaction Report

Programming Aids

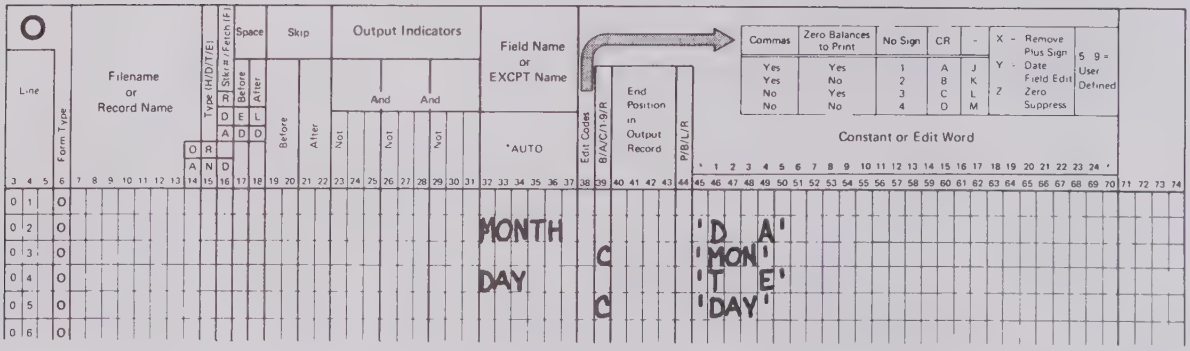
The chart shown in Table 16-1 on page 16-23 should be helpful in determining valid *AUTO output entries depending on the contents of position 39.

The following programming suggestions may be helpful in specific programming situations:

- One column heading can be printed over two or more fields if automatic column spacing is taken into consideration. For example, if the heading DATE is to print over a month field and a day field as follows:

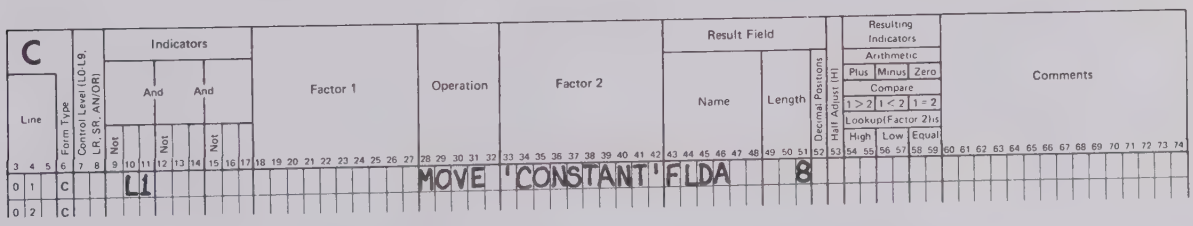


Code the output specifications as follows:



- To print a constant on only the first detail line under a column heading, move the constant to a field in calculation specifications and print that field as shown in Figure 16-14 on page 16-23.
- If group printing is being done and more than one record type is present in the input file, certain precautions must be taken. If a field to be accumulated is present in all record types, but only one record type is to be processed, the correct total is not generated unless additional coding is used. The specifications shown in Figure 16-15 on page 16-24 give incorrect results because the T-*AUTO specification causes an unconditioned ADD subroutine to be generated if a field is to be added. Therefore, QTY is added when indicator 10 is on and when indicator 11 or 12 is on. Figure 16-16 on page 16-25 shows a method of obtaining the correct results.
- Figure 16-17 on page 16-26 shows the specifications for counting records. This method is especially applicable when you want to print a detail list, to take totals by control level, or to eliminate 1's from being listed down the page.

Table 16-1. Valid *AUTO Entries Depending on the Contents of Position 39							
39	**7-22**	**23-31**	**32-37**	**38**	**40-43**	**44**	**45-70**
Blank	Blank	Blank or indicators	Field name	Blank or edit code	Blank or end position	Blank	Blank or column heading
	Blank	Blank or indicators	Blank	Blank	Blank or end position	Blank	Constant
B	Blank	Blank or indicators	Field name	Blank or edit code	Blank or end position	Blank	Blank or column heading
A	Blank	Blank or indicators	Field name	Blank or edit code	Blank or end position	Blank	Blank or column heading
C	Blank	Blank	Blank	Blank	Blank	Blank	Column heading
1-9, R	Blank	Blank	Field name	Blank or edit code	Blank	Blank	Blank or edit word
	Blank	Blank	Blank	Blank	Blank	Blank	Constant



Assume L1 is defined in positions 59 and 60 on input specifications.

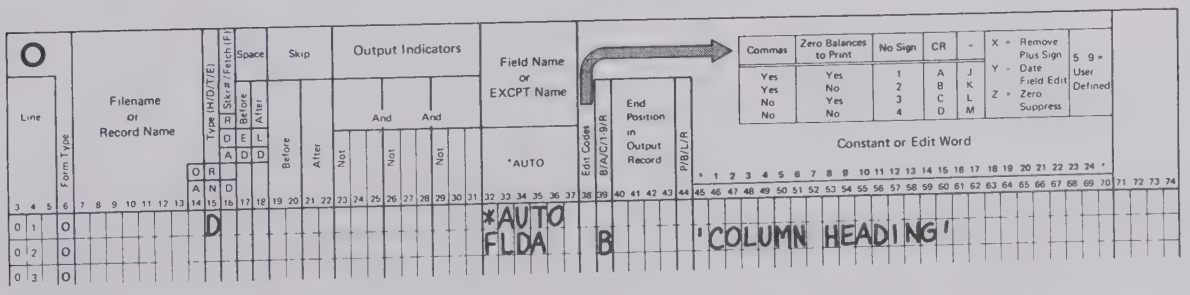
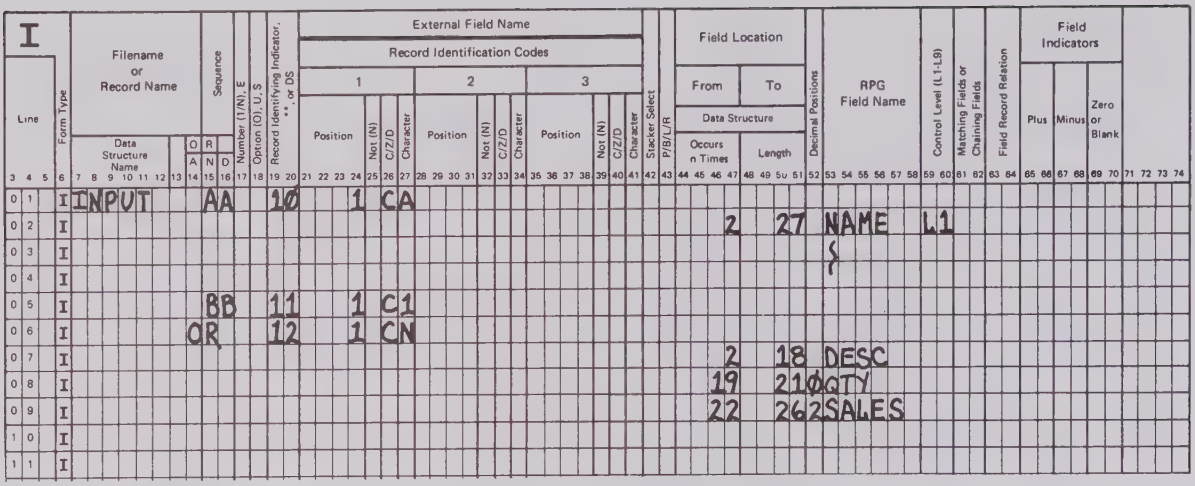


Figure 16-14. Printing a Constant Only on the Detail Line



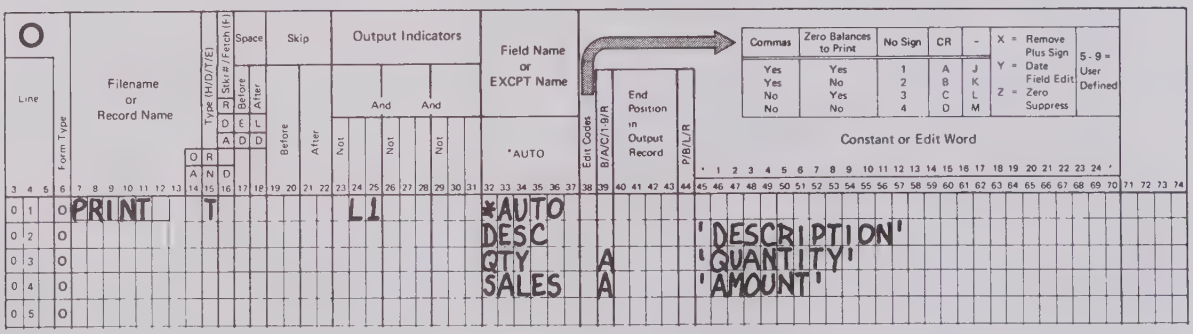


Figure 16-15. Incorrect *AUTO Specifications for More Than One Record Type

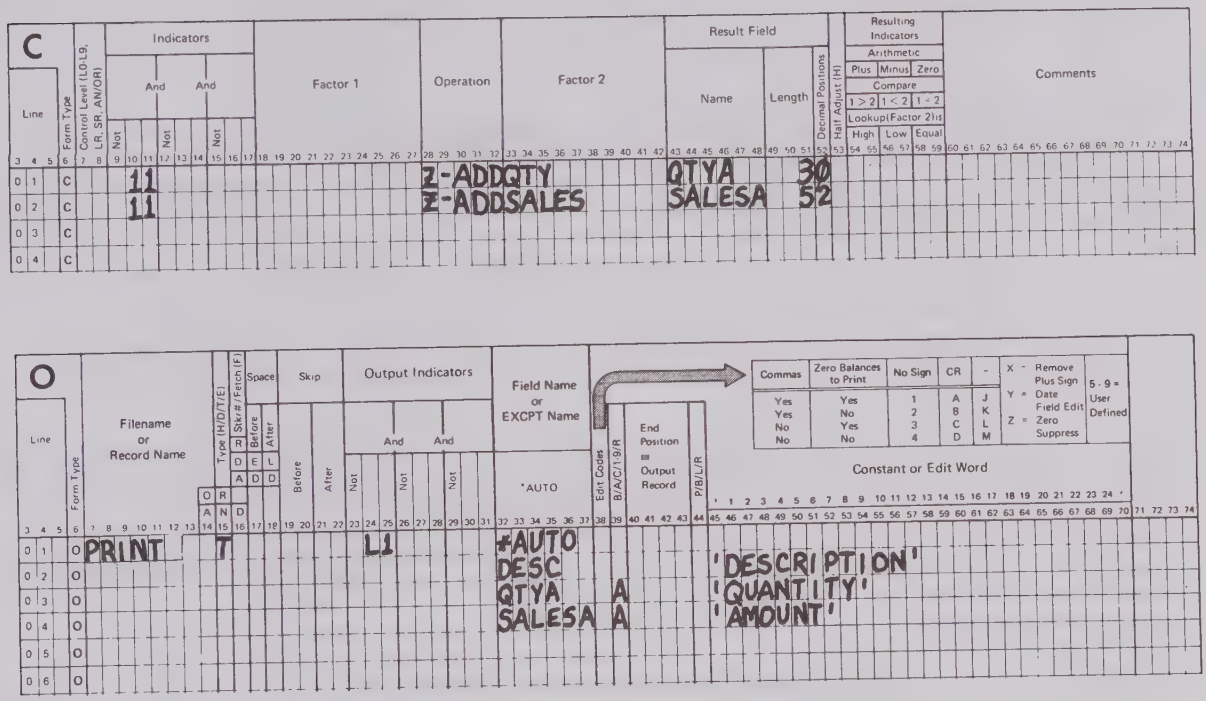
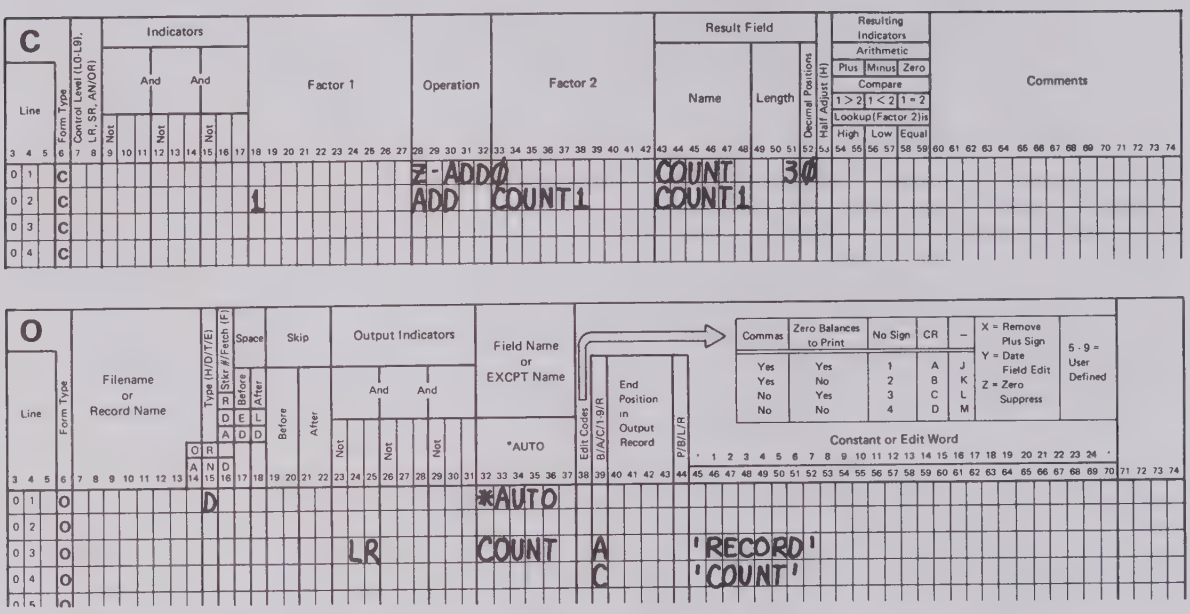


Figure 16-16. Correct *AUTO Specifications for More Than One Record Type



Calculation Specifications

Line 01 This instruction is needed only to define the field COUNT for accumulation.

Line 02 This instruction accumulates the total for the first control level.

Output Specifications

Line 03 This instruction causes the generation of calculation and output specifications for the detail and total lines. The LR conditioning indicator prevents the generated detail calculation from occurring. It also prevents printing at detail time.

Note: If no control levels are specified in the program, a 1 is added to COUNTR rather than to COUNT1 on the calculation specifications.

Figure 16-17. Method of Using *AUTO for Counting Records

Using CRTRPTPGM to Compile an Auto Report Source Program

To compile an RPG/400 source program that includes auto report specifications, you must use the CL command CRTRPTPGM (Create Auto Report Program). RPG/400 program objects are created with the public authority of *CHANGE. You may want to change this authority to maintain greater security on your system.

Auto report does not diagnose all error conditions in the source program. Diagnostics that are produced by the RPG/400 compiler are not duplicated by auto report. If a program cannot be successfully generated because of errors in the auto report specifications, auto report ends. If auto report stops, the escape message RPT 9001 is issued. A CL program can monitor for the escape message by using the CL command MONMSG (Monitor Message).

If an RPG/400 source program is successfully generated and the *NOCOMPILE option is not specified on the CRTRPTPGM command, auto report calls the RPG/400 compiler.

All object names specified on the CRTRPTPGM command use the full naming convention. The length of the name cannot exceed ten characters. See the *Programming: Control Language Reference* for a detailed description of OS/400 object naming rules and for a complete description of OS/400 command syntax.

Using the CRTRPTPGM Command

You call the CRTRPTPGM compiler in three ways:

- Interactively from a display. Type the command CRTRPTPGM and then press F4.
- Using keyword parameters
- Using positional parameters.

See the description of these features of the CRTRPGPGM command in Chapter 3, "Compiling an RPG/400 Program" on page 3-1 for details.

CRTRPTPGM Command

The CRTRPTPGM command is similar to the CRTRPGPGM command described in Chapter 3, "Compiling an RPG/400 Program" on page 3-1 All object names must consist of alphanumeric characters. The first character must always be alphabetic, and the length of the name cannot exceed ten characters.

The CRTRPTPGM command recognizes all the parameters that the CRTRPGPGM command does. Some of these parameters, however, are not used by auto report itself, but are passed on to the RPG/400 compiler. These are the PGM, OPTION,GENOPT, GENLVL, USRPRF, AUT, TEXT, and PHSTRC parameters. The PRTFILE parameter specifies a file that auto report itself uses, and then passes on to the RPG/400 compiler.

The CRTRPTPGM command has the same parameters as the CRTRPGPGM command plus three others: RPTOPT, OUTFILE, and OUTMBR. The description of these parameters follows. The defaults are explained first and are underlined. See "Create RPG III Program (CRTRPGPGM) Command" on page 3-2 for the definition of the other parameters.

The CRTRPTPGM command can be submitted in a batch input stream, entered interactively at a work station, or in a CL program.

RPTOPT

Specifies the options to use when the source program is compiled. Any or all of the following keyword options can be specified in any order. Separate the keywords with a delimiter. The possible values of are:

*NOSOURCE

Do not produce a listing of the auto report source program compile-time errors.

*SOURCE

Produce a listing of the auto report source program compile-time errors.

The acceptable abbreviation for *NOSOURCE is *NOSRC and for *SOURCE is *SRC.

*NOFLOW
Do not write a flow of the major routines run while the auto report source program is compiled.

*FLOW
Write a flow of the major routines run while the auto report source program is compiled.

*COMPILE
Call the RPG/400 compiler after the auto report source statements are processed, and the complete RPG/400 source program is generated.

*NOCOMPILE
Do not call the RPG/400 compiler after the auto report source statements are processed.

*DATE
Include the page number and date on the first *AUTO page heading line.

*NODATE
Do not include the page number and date on the first *AUTO page heading line.

*NOAST
Do not generate asterisk indication for total output lines.

*AST
Generate asterisk indication for total output lines.

*NOSECLVL
Do not print second-level text on the line following the first level message text.

*SECLVL
Print second-level text on the line following the first level message text.

OUTFILE
Specifies the name of the file where the complete RPG/400 source program is to be placed and the library in which the file is located. The file is also used as the source input file to the RPG/400 compiler unless the RPTOPT parameter value *NOCOMPILE is specified.

*LIBL
If a library name is not specified, the IBM-supplied source file, QRPGSRC, contains the RPG/400 source program to be compiled.

*CURLIB
The name of the current library. If a current library is not specified, QGPL is the current library.

library-name
Enter the name of the library in which the file is located.

*NONE
Create a file in QTEMP to pass the generated RPG/400 source to the RPG/400 compiler.

file-name:
Enter the name of the file to contain the complete RPG/400 source program.

OUTMBR
Specifies the name of the member of the file that will contain the output from auto report.

*NONE
Use the first member created in or added to the file as the member name.

file-member-name
Enter the name of the member that is to contain the output of auto report.

Examples of Using Auto Report

Examples 1 through 4 explain how auto report is used to generate report page headings and such output specifications as column headings, detail lines, and total lines. Examples 5 and 6 illustrate the use of the auto report copy function to copy specifications from a source file member and to modify copied specifications for a particular job.

EXAMPLE 1-Sales Report

Problem

Produce the sales report shown below using the *AUTO page headings and *AUTO output functions of auto report.

*AUTO Page Headings | *AUTO Output

Procedure

1. Code normal RPG file description and input specifications for the job.
2. Code *AUTO page headings to produce a one-line page heading that includes date and page number.
3. Code *AUTO output to produce one-line column headings, detail report lines, and final totals.

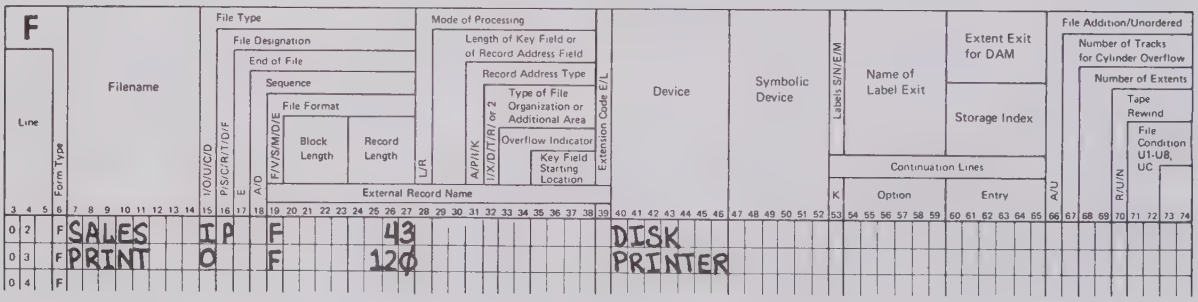
Examples of Using Auto Report

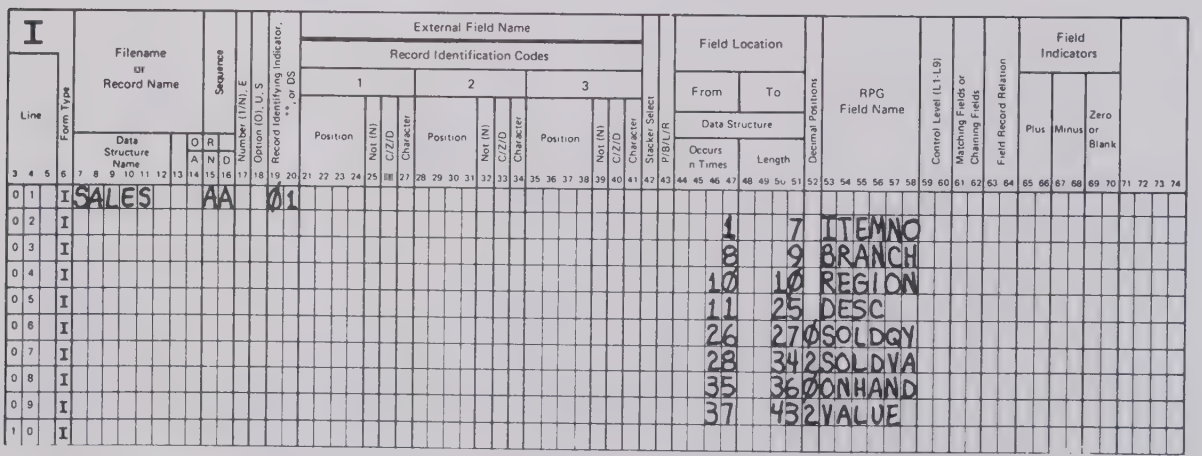
Letters refer to fields on the following page.

```
10/26/80                    SALES REPORT FOR ANY CO.                    PAGE  1
   C          B          A          D          E          F          G          H
REGION    BRANCH     ITEM     DESCRIPTION    SALES     AMOUNT    ON-HAND     VALUE

  1         17      AG7701T   2-TON TRUCK        5   25,000.00          2   10,000.00
  1         17      AG77055   PICK-UP           10   20,000.00          1    2,000.00
  1         17      AP6545B   CAMPER             2    8,000.00
  1         22      AG7701T   2-TON TRUCK        2   10,000.00          1    5,000.00
  1         22      AG77055   PICK-UP            4    8,000.00          1    2,000.00
  3         25      AG6545B   CAMPER            10   40,000.00          5   20,000.00
  3         25      AP6549P   1/4 TON TRUCK     20   30,000.00          6    9,000.00

                                                    141,000.00              48,000.00 *
```

1 Code RPG/400 file description and input specifications.

File Description Specifications

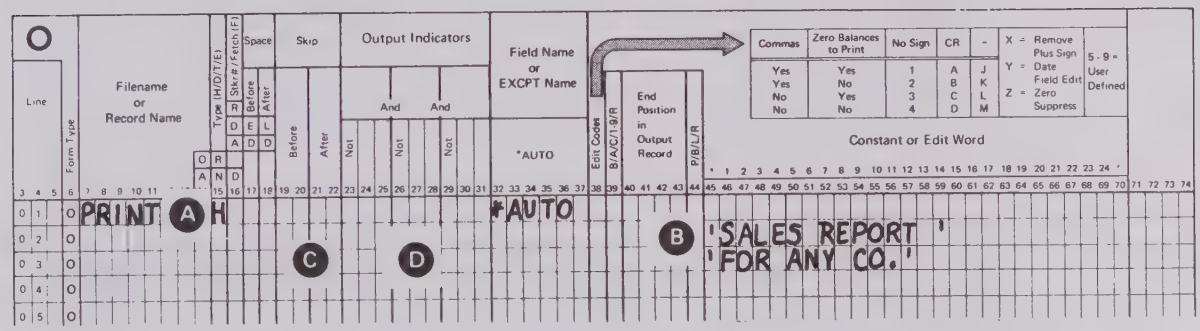




	Field Name	Contents
A	ITEMNO	Item number

	Field Name	Contents
B	BRANCH	Number of the branch office where the item was sold
C	REGION	Sales region in which the branch office is located
D	DESC	Description of the sales item
E	SOLDQY	Quantity of the item sold
F	SOLDVA	Total value of the items sold
G	ONHAND	Quantity of the item remaining on hand
H	VALUE	Total value of the items remaining on hand.

2 Code *AUTO page heading specifications.



A Enter an H in position 15 and *AUTO in positions 32 through 36 to request an auto report page heading. Up to five page heading lines can be described. The system date is printed on the left and the page number on the right of the first heading line on each page. To suppress the date and page, enter an N in position 27 of the auto report option specifications or use the *NODATE option on the RPTOPT parameter in the CRTRPTPGM command.

B The title information is centered by auto report; do not enter end positions in positions 40 through 43. Fields and array/table elements can also be used.

C When space and skip entries (positions 17 through 22) are left blank, skip to line 06 is assumed for the first heading line; single spacing is done between heading lines, double spacing after the last heading line. (See "Example 4" for an example of multiple page heading lines.)

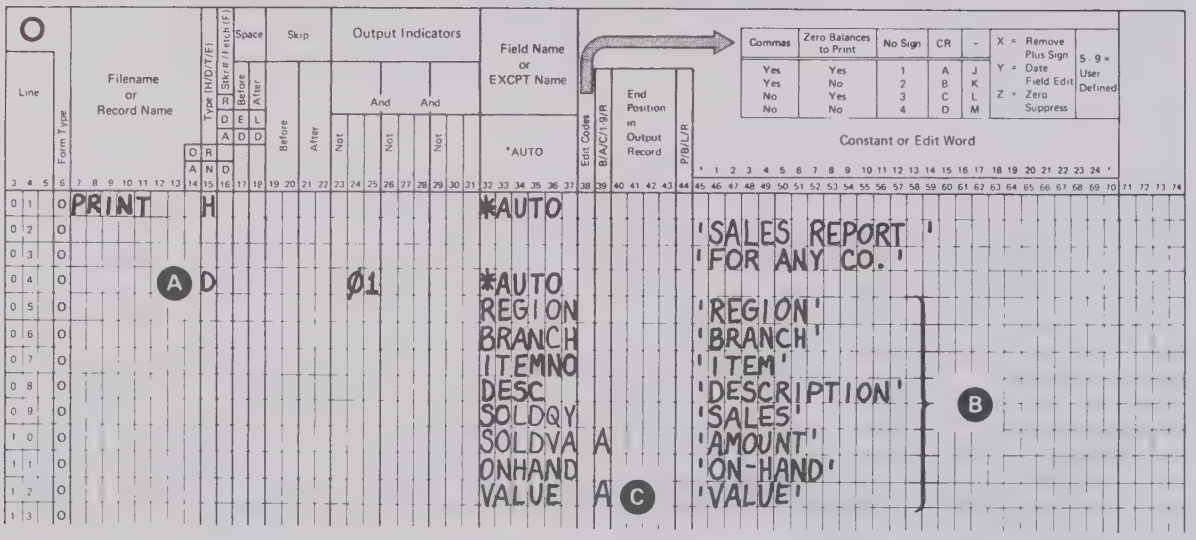
D When output indicators (positions 23 through 31) are left blank, auto report page headings are printed on each page (conditioned by 1P or overflow). If no overflow indicator is defined for the printer file, auto report assigns an unused overflow indicator to the printer line.

C Line 06

Blank line

```
10/26/80 A                SALES REPORT FOR ANY CO. B          A PAGE    1
REGION   BRANCH   ITEM    DESCRIPTION    SALES      AMOUNT ON-HAND      VALUE

  1        17     AG7701T  2-TON TRUCK      5     25,000.00      2     10,000.00
  1        17     AG77055  PICK-UP         10     20,000.00      1      2,000.00
  1        17     AP6545B  CAMPER           2      8,000.00
  1        22     AG7701T  2-TON TRUCK      2     10,000.00      1      5,000.00
  1        22     AG77055  PICK-UP          4      8,000.00      1      2,000.00
  3        25     AG6545B  CAMPER          10     40,000.00      5     20,000.00
  3        25     AP6549P  1/4 TON TRUCK   20     30,000.00      6      9,000.00

                                          141,000.00            48,000.00 *
```

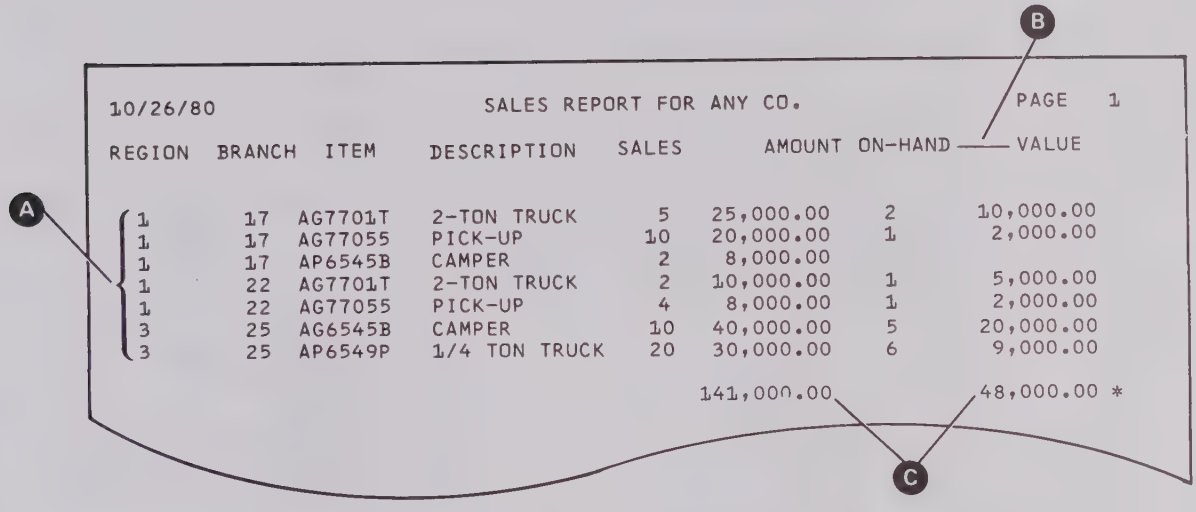
3 Code *AUTO output specifications to produce:



A Enter D in position 15 and *AUTO in positions 32 through 36 to describe an auto report with detail lines. The record identifying indicator 01 conditions printing of the detail lines.

B Column headings are entered on the same line as the fields over which they appear in the report.

C Enter an A in position 39 to cause fields to be accumulated. Auto report generates (1) total fields and calculations to accumulate the totals, and (2) total output specifications to print the totals.



Auto report formats the report so that column headings and data are neatly spaced and centered on each other.

All numeric fields for which a blank, B, or A is specified in position 39 are edited by the K edit code unless a different edit code is specified.

EXAMPLE 2-Sales Report with Three Levels of Totals

Problem

Expand sales report from *Example 1* to include three levels of totals:

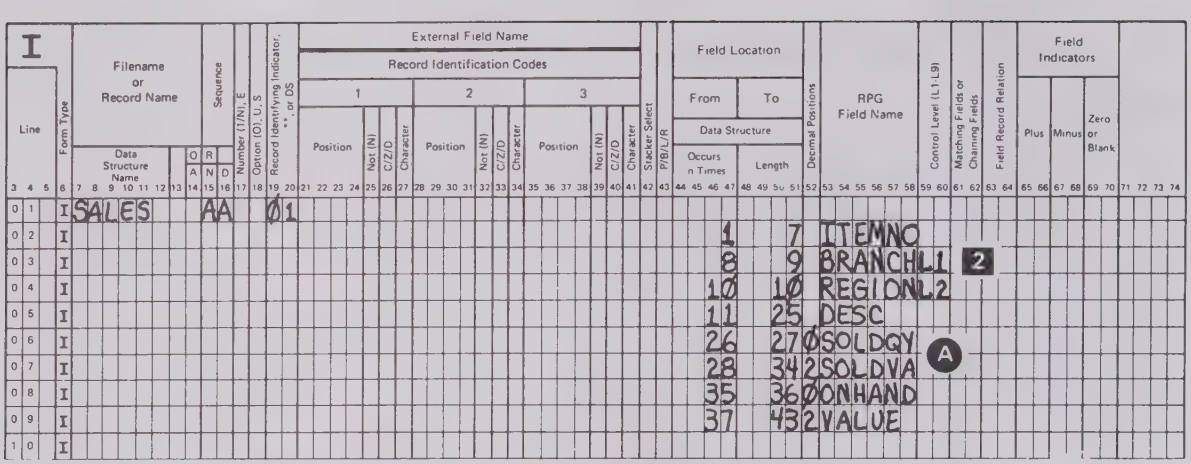
1. Total for each branch
2. Total for each region
3. Final total

***AUTO Output**

Procedure

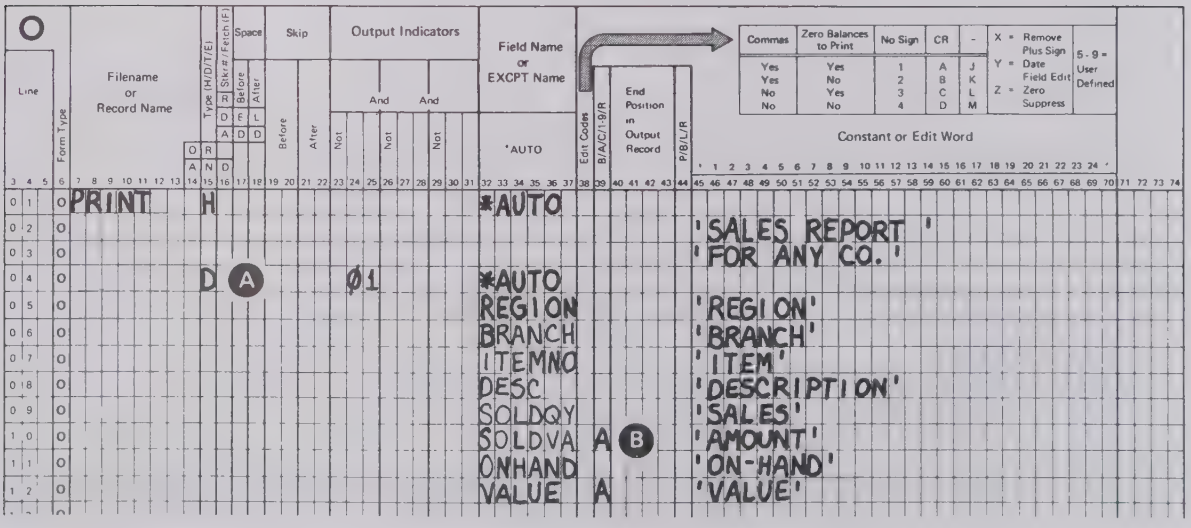
1. Code file description and *AUTO specifications as in *Example 1.*
2. Add control level indicators to the input fields BRANCH and REGION.

Note: The *AUTO output function can also be used to produce a group printed report. See "Group Printing" on page 16-1 for a discussion and examples of group printing.



Line	Form Type	Filename or Record Name	Sequence	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, *, or DS	External Field Name	Record Identification Codes	Stacker Select	P/B/L/R	Field Location From	Field Location To	Data Structure Occurs n Times	Data Structure Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank
01	I	SALES	AA			Ø1																
02	I										1	7				ITEMNO						
03	I										8	9				BRANCHL1	2					
04	I										10	10				REGIONL2						
05	I										11	25				DESC						
06	I										26	27			Ø	SOLDQY						
07	I										28	34	2			SOLDVA	A					
08	I										35	36	Ø			ONHAND						
09	I										37	43	2			VALUE						
10	I																					

A Because two control levels are defined, the SOLDVA and VALUE fields (see following page) are accumulated to two levels of totals (branch and region) and a final total (LR).



Line	Form Type	Filename or Record Name	Type (H/D/T/E)	Skip Before	Space After	Skip	Output Indicators	Field Name or EXCPT Name	Edit Codes	B/A/C/1-9/R	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	PRINT	H					*AUTO					'SALES REPORT '
02	O												'FOR ANY CO.'
03	O												
04	O		D				Ø1	*AUTO					
05	O							REGION					'REGION'
06	O							BRANCH					'BRANCH'
07	O							ITEMNO					'ITEM'
08	O							DESC					'DESCRIPTION'
09	O							SOLDQY					'SALES'
10	O							SOLDVA	A				'AMOUNT'
11	O							ONHAND					'ON-HAND'
12	O							VALUE	A				'VALUE'

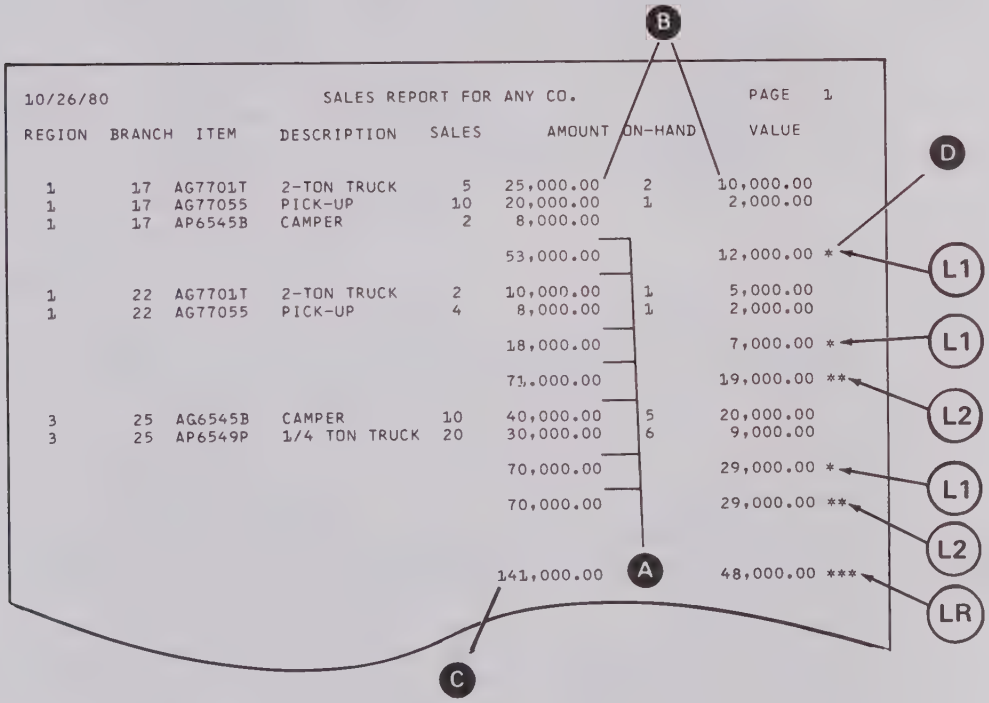
Commas: Yes, Yes, No, No
Zero Balances to Print: Yes, No, Yes, No
No Sign: 1, 2, 3, 4
CR: A, B, C, D
-: J, K, L, M
X = Remove Plus Sign; Y = Date Field Edit; Z = Zero Suppress
5-9 = User Defined

A Auto report places a blank line after each total line and an additional blank line before the lowest level total and before the final total. If you enter spacing and skipping values on the D-*AUTO specification, they apply to the detail print line only.

B As in "Example 1", an A in position 39 of the output specification causes SOLDVA and VALUE to be accumulated.

C Total fields are always two positions longer with the same number of decimal positions as the original fields.

D Auto report prints asterisks (*) to the right of the generated total lines to aid in identifying them. If you want to suppress the asterisks, enter N in position 28 of the auto report option specifications or use the *NOAST option on the RPTOPT parameter in the CRTRPTPGM command.



EXAMPLE 3-Sales Report with Group Indication

Problem

Expand the sales report from *Examples 1* and *2* to contain:

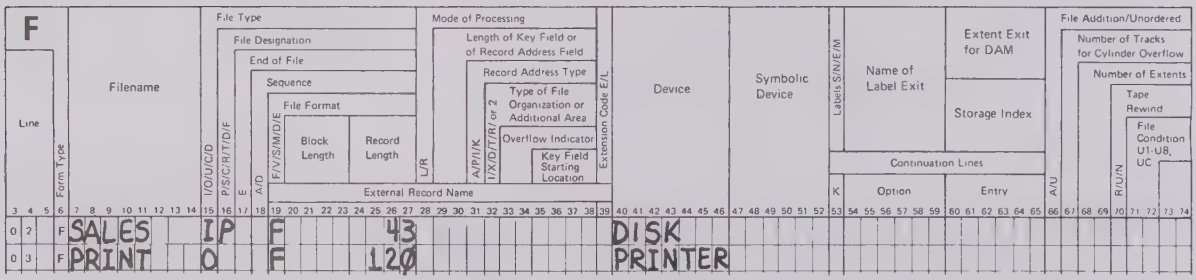
- **A** Group indication for REGION and BRANCH fields
- **B** Second column heading line
- **C** Literal (constant) on the final total line

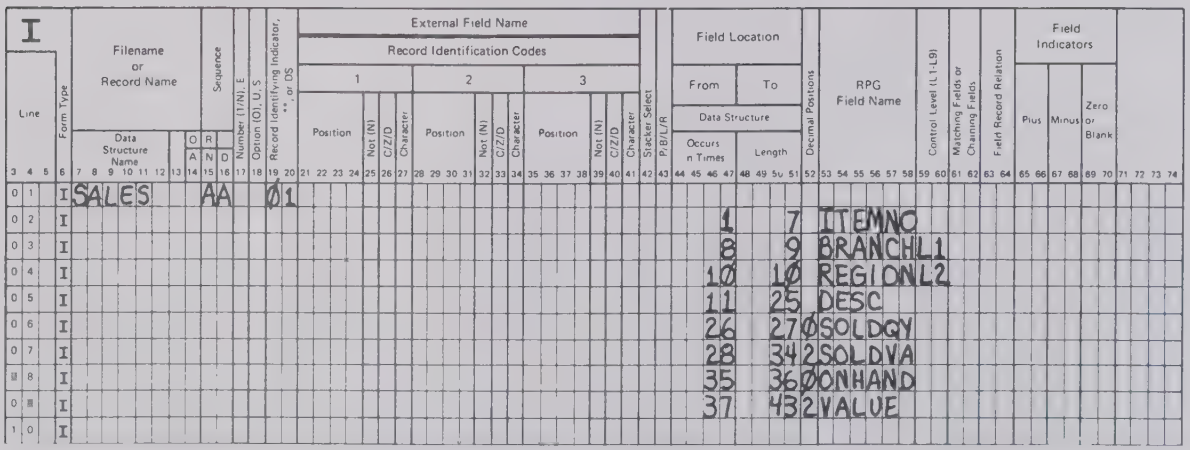
***AUTO Output**

Procedure

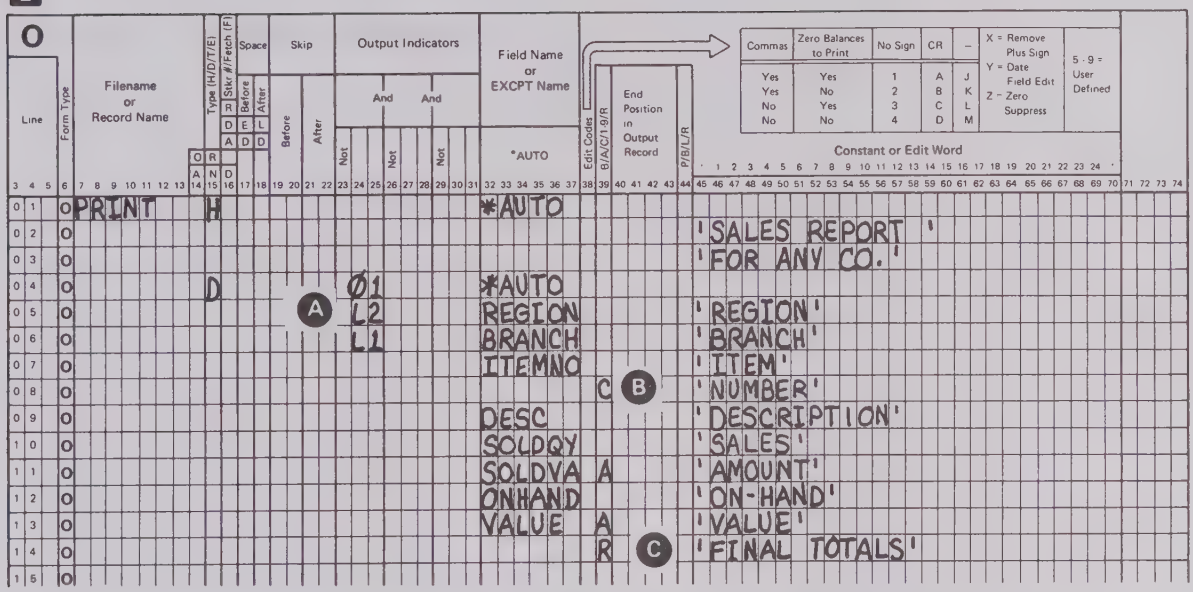
1. Code file description and input specifications as for *Example 2.*
2. Code *AUTO output with:
   - **A** Output indicator on field description specifications.
   - **B** C in position 39 and a constant in positions 45 through 70.
   - **C** R in position 39 and a constant in positions 45 through 70.

1





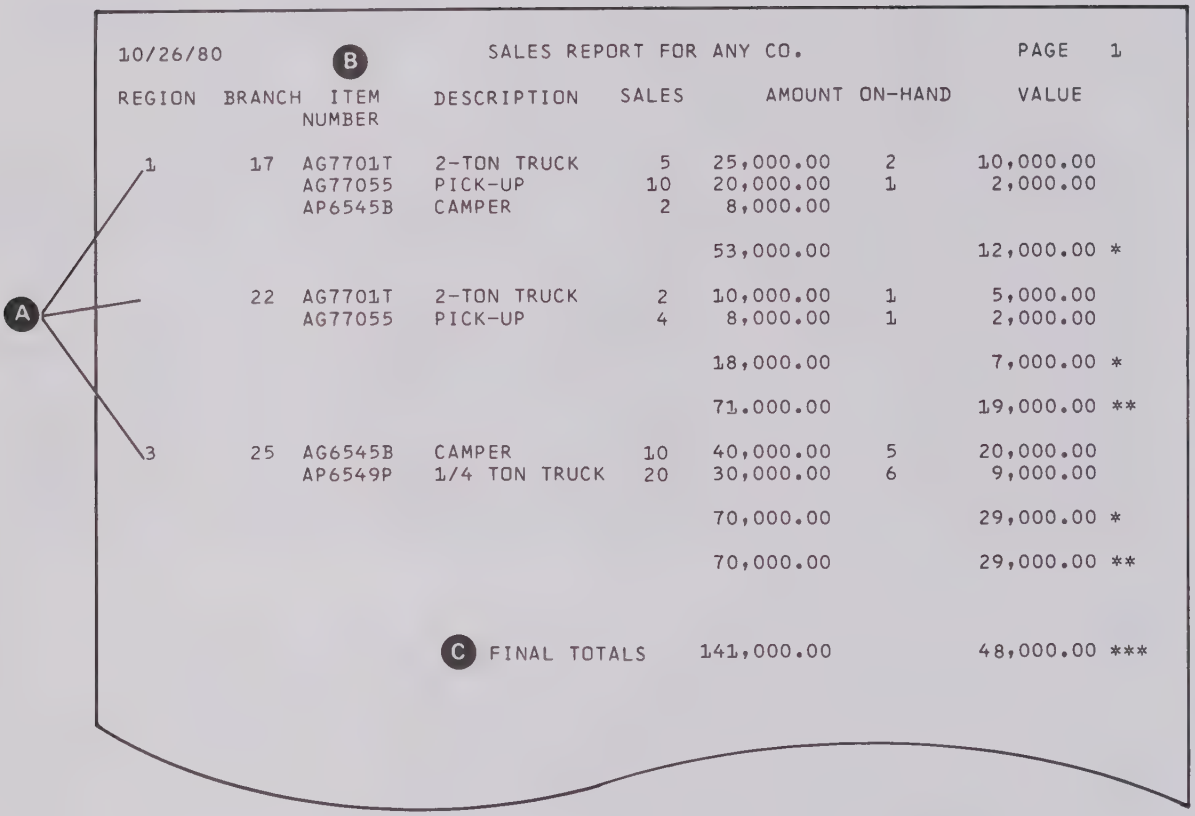
2



A Output indicators can be used on field description specifications. In this example, control level indicators condition BRANCH and REGION so that they are printed only for the first record of the corresponding control group. This print suppressing of common fields (group indication) reduces repetitive information.

B One or two additional column heading lines can be specified by a C entry in position 39 with the heading information in positions 45 through 70.

C The literal FINAL TOTALS makes that line easy to find. To specify information to appear on the final total line, enter R in position 39 with a constant in positions 45 through 70 or a field name/table name/indexed array name in positions 32 through 37. The information is printed two spaces to the left of the leftmost total on the line. If more than one such specification is used, the constants and fields are printed from left to right in the order they are specified in the program.



10/26/80 (B) SALES REPORT FOR ANY CO. PAGE 1

REGION	BRANCH	ITEM NUMBER	DESCRIPTION	SALES	AMOUNT	ON-HAND	VALUE	
1	17	AG7701T	2-TON TRUCK	5	25,000.00	2	10,000.00	
		AG77055	PICK-UP	10	20,000.00	1	2,000.00	
		AP6545B	CAMPER	2	8,000.00			
					53,000.00		12,000.00	*
	22	AG7701T	2-TON TRUCK	2	10,000.00	1	5,000.00	
		AG77055	PICK-UP	4	8,000.00	1	2,000.00	
					18,000.00		7,000.00	*
					71.000.00		19,000.00	**
3	25	AG6545B	CAMPER	10	40,000.00	5	20,000.00	
		AP6549P	1/4 TON TRUCK	20	30,000.00	6	9,000.00	
					70,000.00		29,000.00	*
					70,000.00		29,000.00	**
			(C) FINAL TOTALS		141,000.00		48,000.00	***

EXAMPLE 4-Sales Report with Cross-Column Totals

Problem

Expand the sales report from *Examples 1, 2,* and *3* to include a cross-totals column and:

- (A) A new report page for each region
- (B) Two heading lines on each page
- (C) A field in a page heading line
- (D) Identification of branch and region totals

***AUTO Page Headings** ***AUTO Output**

Procedure

1. Code file description and input specifications as in *Example 3;* add an overflow indicator to the printer file.
2. Code RPG calculation specifications for cross-total.
3. Code *AUTO specifications:
   - (A) Output indicators on page heading specifications
   - (B) Two heading lines per page
   - (C) Use of a field in an *AUTO page heading specification
   - (D) Fields and constants on L1 through L9 total lines (1 through 9 in position 39)

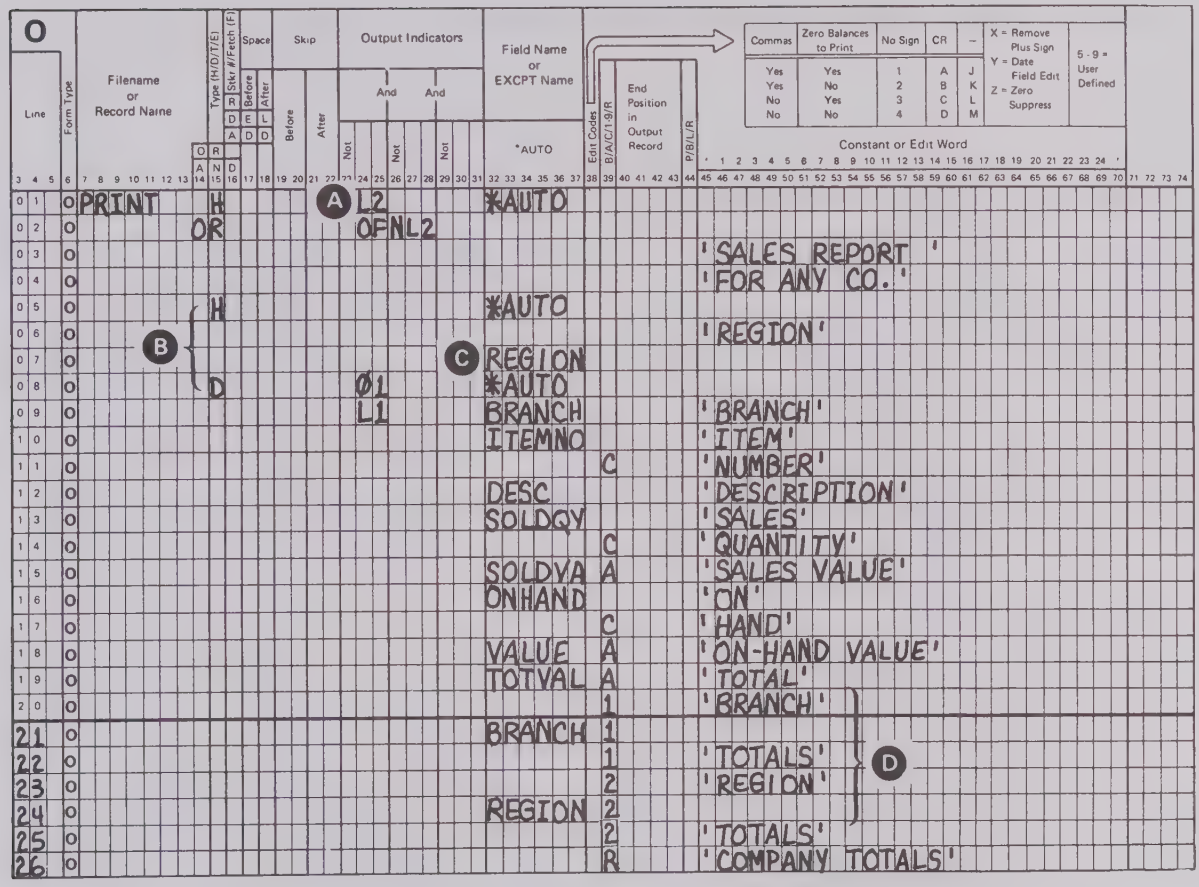
```
11/11/80                          SALES REPORT FOR ANY CO.                          PAGE     1
                                        REGION 1                                      2
BRANCH   ITEM NUMBER   DESCRIPTION      SALES QUANTITY   SALES VALUE   ON HAND   ON-HAND VALUE   TOTAL
  17     AG7701T       2-TON TRUCK              5          25,000.00       2        10,000.00     35,000.00
         AG7705S       PICK-UP                 10          20,000.00       1         2,000.00     22,000.00
         AP6545B       CAMPER                   2           8,000.00                               8,000.00
                       BRANCH 17 TOTALS                     53,000.00                12,000.00     65,000.00 *
  22     AG7701T       2-TON TRUCK              2          10,000.00       1         5,000.00     15,000.00
         AG7705S       PICK-UP                  4           8,000.00       1         2,000.00     10,000.00
                       BRANCH 22 TOTALS                     18,000.00                7,000.00     25,000.00 *
                       REGION 1 TOTALS                      71,000.00                19,000.00    90,000.00 **
```

```
                                                                                      A
11/11/80                          SALES REPORT FOR ANY CO.                          PAGE     2
                                        REGION 3
BRANCH   ITEM NUMBER   DESCRIPTION      SALES QUANTITY   SALES VALUE   ON HAND   ON-HAND VALUE   TOTAL
  25     AG6545B       CAMPER                  10          40,000.00       5        20,000.00     60,000.00
         AP6549P       1/4 TON TRUCK           20          30,000.00       6         9,000.00     39,000.00
                       BRANCH 25 TOTALS                     70,000.00                29,000.00     99,000.00 *
                       REGION 3 TOTALS                      70,000.00                29,000.00     99,000.00 **
                       COMPANY TOTALS                      141,000.00                48,000.00    189,000.00 ***
```

Note: Compare matching letters (B) on this and the following pages to see the auto report coding to obtain this report.

2 RPG/400 calculations can be among the input statements for auto report. This specification calculates a cross-total of the sales and on-hand values. The placement of the calculation in relation to calculations generated by auto report is described under "Generated Specifications" on page 16-16.

C	Line	Form Type	Control Level (L0-L9), LR, SR, AN/OR	Indicators: And / And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Result Field: Length	Result Field: Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic: Plus / Minus / Zero	Resulting Indicators: Compare: 1>2 / 1<2 / 1=2	Resulting Indicators: Lookup(Factor 2)is: High / Low / Equal	Comments
	01	C		Ø1	SOLDVA	ADD	VALUE	TOTVAL	82						
	02	C													



A The headings are printed on a new page when the region number changes (L2) or when overflow occurs (OF). (OF must be defined for the printer file in file description specifications).

B A second auto report page heading is specified. Because spacing is not specified, space-one is done after the first and space-two after the second. Because no output indicators are specified, the second heading is conditioned like the first.

C The contents of the REGION field are printed on the second page heading.

D Fields and constants can be printed on generated total lines if you enter the number of the control level in position 39.

EXAMPLE 5-Sales Report using copied Specifications

Problem

Use the copy function to obtain specifications for the sales report below (same as in *Example 1*).

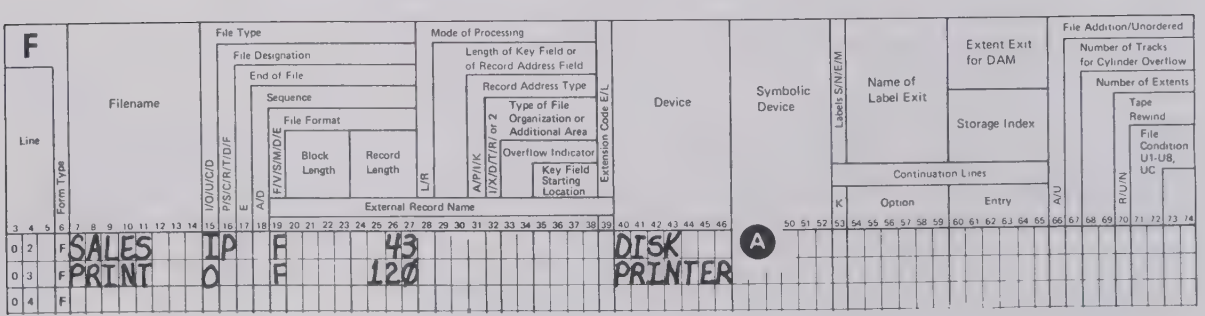
COPY

Procedure

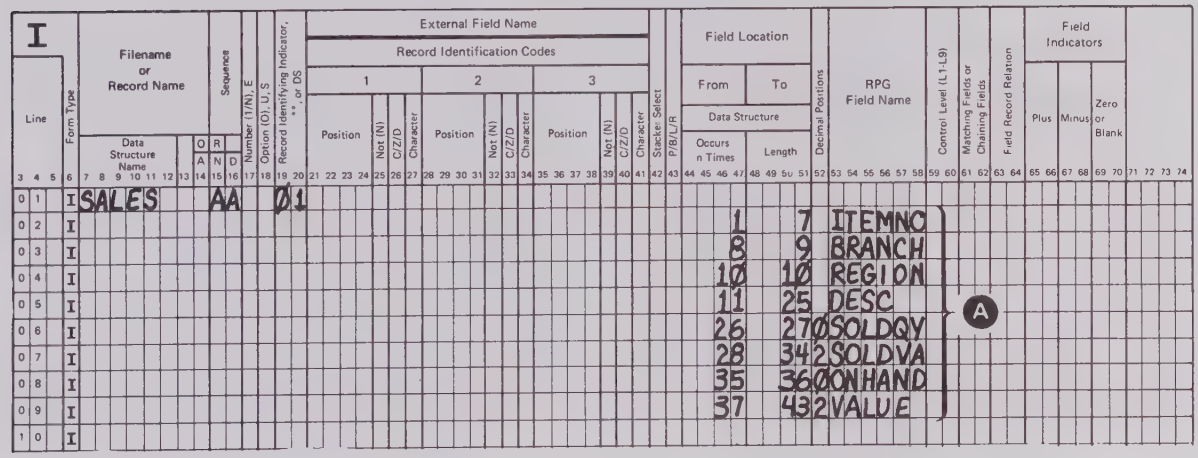
1. Save the file description and input specifications for the SALES file in a source file member.
2. Code the /COPY statement in the specifications for auto report.

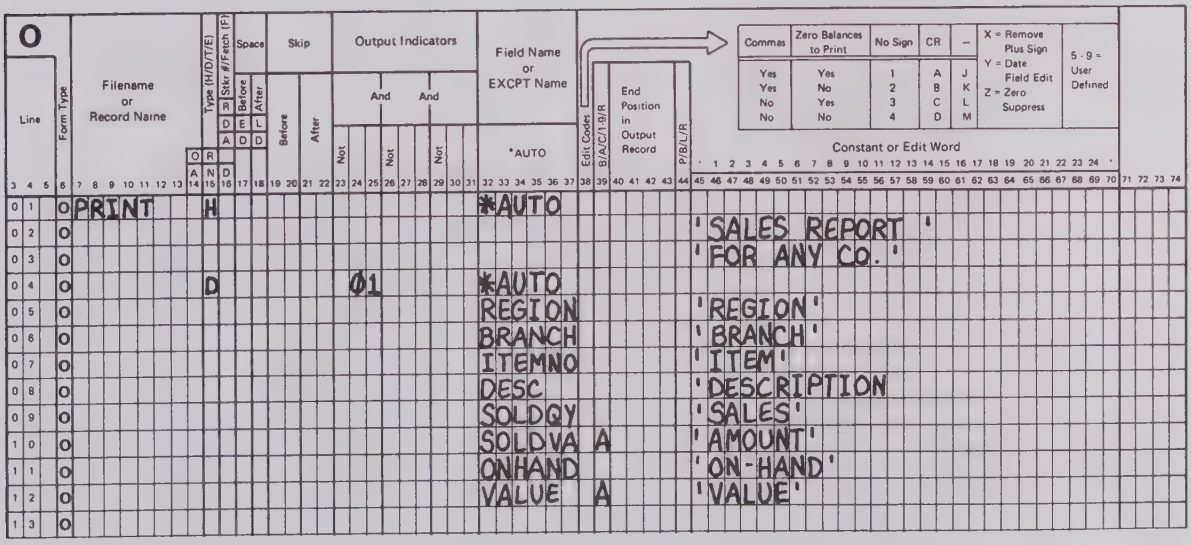
```
10/26/80                    SALES REPORT FOR ANY CO.                    PAGE   1

REGION  BRANCH  ITEM     DESCRIPTION   SALES     AMOUNT  ON-HAND     VALUE

  1       17    AG7701T  2-TON TRUCK       5   25,000.00      2     10,000.00
  1       17    AG77055  PICK-UP          10   20,000.00      1      2,000.00
  1       17    AP6545B  CAMPER            2    8,000.00
  1       22    AG7701T  2-TON TRUCK       2   10,000.00      1      5,000.00
  1       22    AG77055  PICK-UP           4    8,000.00      1      2,000.00
  3       25    AG6545B  CAMPER           10   40,000.00      5     20,000.00
  3       25    AP6549P  1/4 TON TRUCK    20   30,000.00      6      9,000.00

                                          141,000.00           48,000.00 *
```

1 Specifications for the SALES file are stored in a source file member.

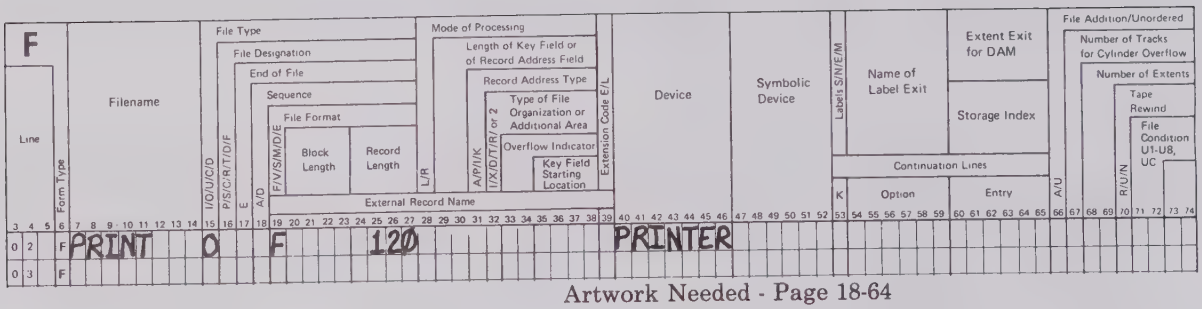


A These specifications could be replaced by a single statement as shown on the following page.

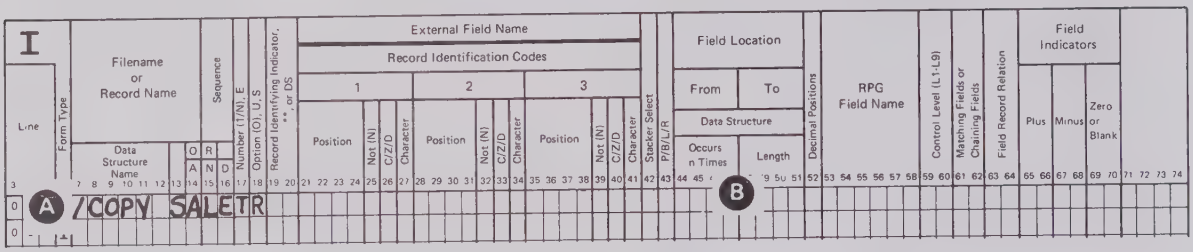




2 Code the /COPY statement to include the file description and input specifications. (For a detailed description of the copy function, see "/COPY Statement Specifications" on page 16-4.)



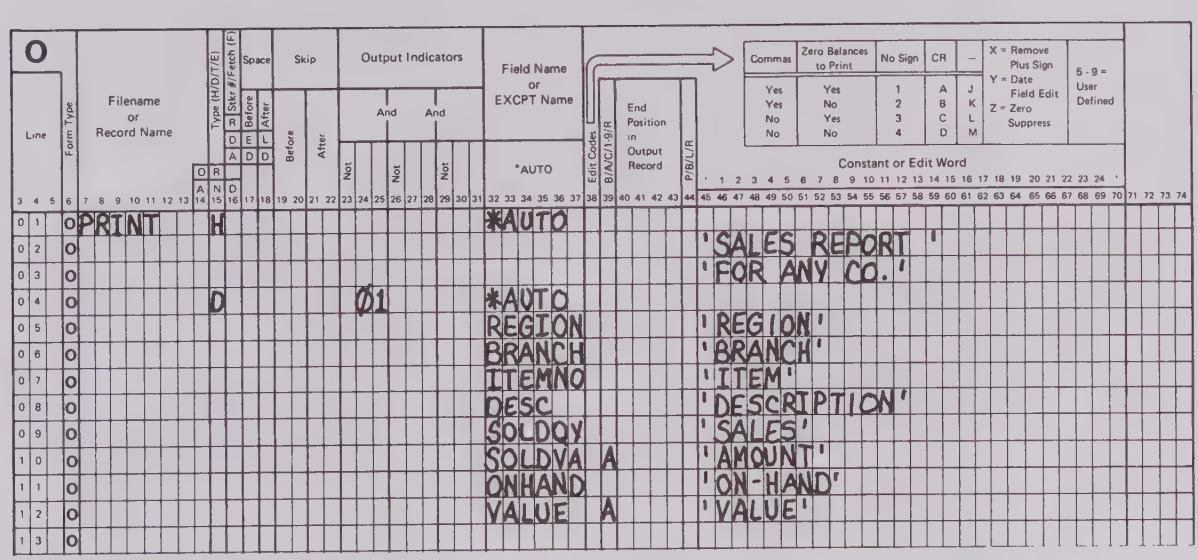
Artwork Needed - Page 18-64



A Position 6 of a /COPY statement must not contain a U or an H.

B The /COPY statement copies file description and input specifications for the SALES file from the member named SALETR.

The /COPY statement can appear anywhere among the auto report specifications following the auto report option statement and preceding array and table input records. It is convenient to code the /COPY on the input specifications when you want to override copied input specifications, as in "Example 6". After specifications are copied, all specifications are sorted into the order required by the RPG/400 compiler.



Line	Form Type	Filename or Record Name	Type (H/D/T/E)	Space	Skip	Output Indicators	Field Name or EXCPT Name	Edit Codes	End Position in Output Record	P/B/L/R	Constant or Edit Word
01	O	PRINT	H				*AUTO				'SALES REPORT '
02	O										'FOR ANY CO.'
03	O										
04	O		D			01	*AUTO				
05	O						REGION				'REGION'
06	O						BRANCH				'BRANCH'
07	O						ITEMNO				'ITEM'
08	O						DESC				'DESCRIPTION'
09	O						SOLDQY				'SALES'
10	O						SOLDVA	A			'AMOUNT'
11	O						ONHAND				'ON-HAND'
12	O						VALUE	A			'VALUE'
13	O										

Commas	Zero Balances to Print	No Sign	CR	–
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

5 - 9 = User Defined

EXAMPLE 6-Override Copied Input Specifications

Problem

Override copied input specifications to produce a report (below) that includes subtotals for branch and region.

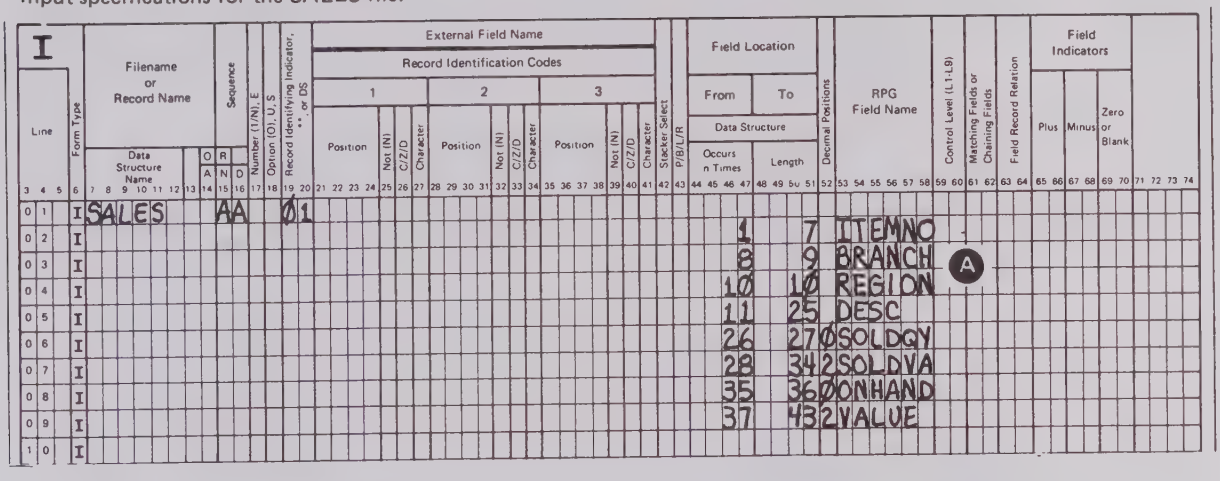
COPY

Procedure

1. Save specifications for the SALES file, as in *Example 5.*
2. Code the /COPY statement.
3. Code /COPY modifier statements to add control level indicators to BRANCH and REGION fields on copied specifications.

```
10/26/80                    SALES REPORT FOR ANY CO.                    PAGE   1

REGION  BRANCH  ITEM     DESCRIPTION    SALES      AMOUNT  ON-HAND      VALUE

  1       17    AG7701T  2-TON TRUCK        5   25,000.00        2   10,000.00
  1       17    AG77055  PICK-UP           10   20,000.00        1    2,000.00
  1       17    AP6545B  CAMPER             2    8,000.00

                                            53,000.00            12,000.00 *

  1       22    AG7701T  2-TON TRUCK        2   10,000.00        1    5,000.00
  1       22    AG77055  PICK-UP            4    8,000.00        1    2,000.00

                                            18,000.00             7,000.00 *

                                            71,000.00            19,000.00 **

  3       25    AG6545B  CAMPER            10   40,000.00        5   20,000.00
  3       25    AP6549P  1/4 TON TRUCK     20   30,000.00        6    ?,000.00
```

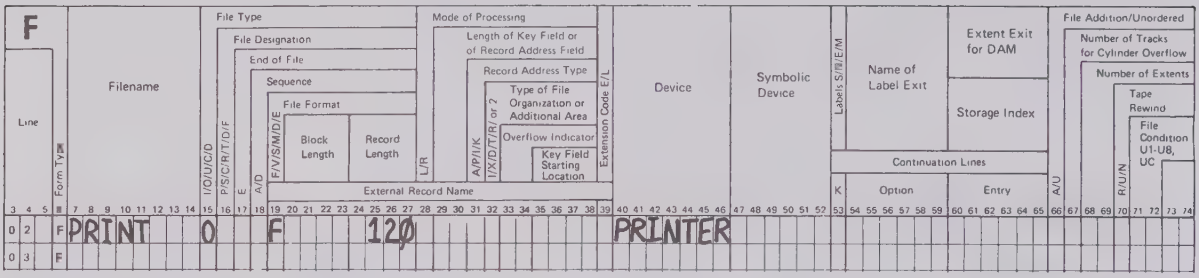
Input specifications for the SALES file.

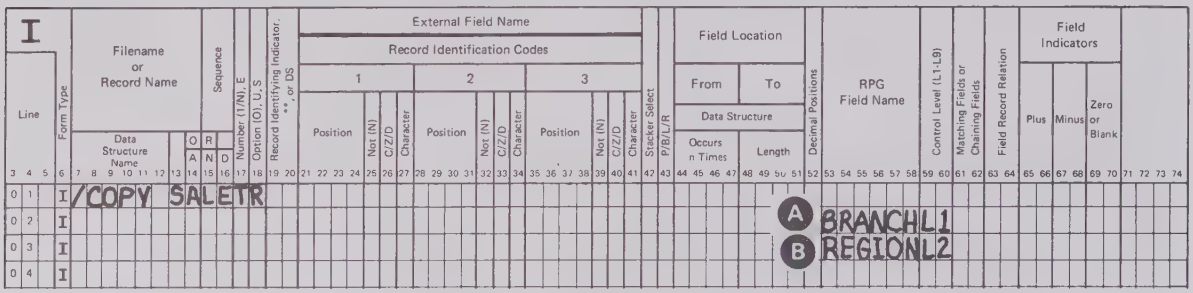


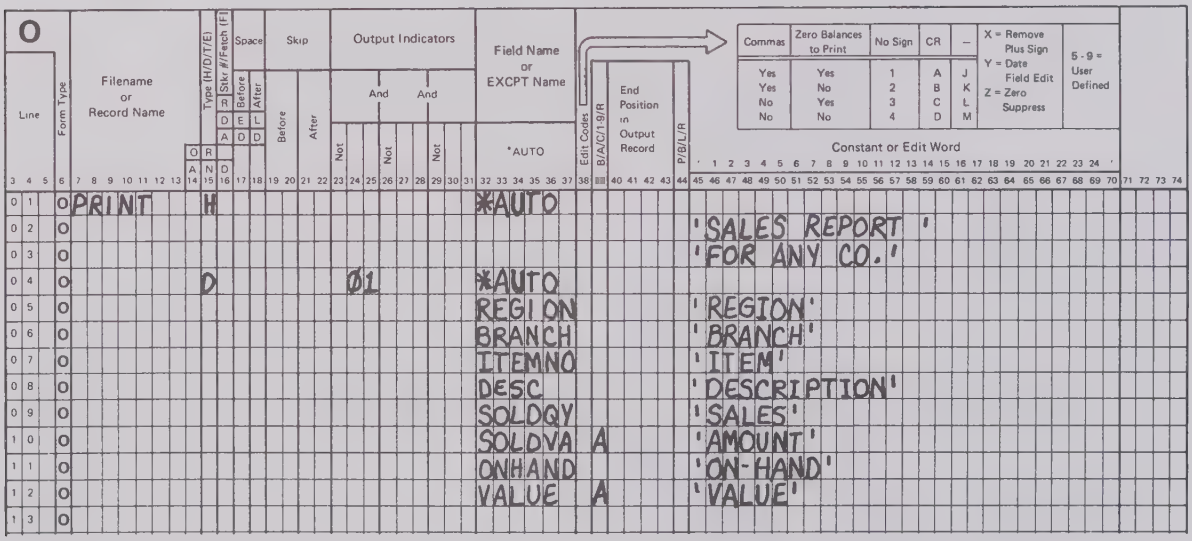
Line	Form Type	Filename or Record Name	Data Structure Name	O A	R N	D	Sequence	Number (1/N), E	Option (O), U, S	Record Identifying Indicator, *, or DS	External Field Name	Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Stacker Select	P/B/L/R	Field Location From	Field Location To	Data Structure Occurs n Times	Data Structure Length	Decimal Positions	RPG Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank
01	I	SALES			AA					01																1	7				ITEMNO						
02	I																									8	9				BRANCH	A					
03	I																									10	10				REGION						
04	I																									11	25				DESC						
05	I																									26	27	0			SOLDQY						
06	I																									28	34	2			SOLDVA						
07	I																									35	36	0			ONHAND						
08	I																									37	43	2			VALUE						
09	I																																				
10	I																																				

A To produce a report that has subtotals for branch and region, L1 must be assigned to BRANCH and L2 to REGION as the specifications are copied from the source file member.

2 and **3** Code /COPY and modifier statements. As a result of the modifier statements, three levels of totals are accumulated for the SOLDVA and VALUE fields (L1, L2 and LR).







A Entries on the modifier statements override the corresponding entries in the copied specifications.

B The field names, BRANCH and REGION, identify the input field specifications that are to be modified.

Saved file description or input specifications are overridden as follows (see "/COPY Statement Specifications" on page 16-4 for examples):

- Entries in a modifier statement override corresponding entries in a copied file description or input field specification.
- Blank entries in a modifier statement remain unchanged in a copied specification.
- Ampersand (&) in the leftmost position of an entry in the modifier statement sets the entry to blanks in the copied specification.
- New fields can be added to input specifications by new input field specifications added as modifier statements.
- Modifier statements do not change the saved specifications. The modification is only for the program into which the specifications are copied.

Appendix A. RPG Compiler and Auto Report Program Service Information

This appendix is provided for RPG/400 service personnel to use when investigating RPG/400 problems and provides the following information:

- Compiler overview
- Compiler debugging options
- IRP layout
- Auto report program overview.

RPG/400 programmers can also use this information to investigate RPG/400 problems on their own before or instead of calling for service.

Compiler Overview

This section provides the following compiler information:

- How the compiler works
- Compiler phase descriptions
- Major compiler data area descriptions
- Compiler error message organization.

Figure A-1 on page A-2 summarizes how an RPG/400 source program is compiled into a (encapsulated) program object.

Intermediate text, which is output from step 1 in Figure A-1 on page A-2, is a representation of RPG/400 source statements that is created by compiler phases and exists only while they are running. This text can be dynamically listed with the ITDUMP parameter of the CL command CRTRPGPGM (Create RPG/400 Program) or can be listed at the completion of any compiler phase with the SNPDUMP parameter of the CL command CRTRPGPGM. Refer to "Compiler Debugging Options" on page A-5 for explanations of these parameters and examples of intermediate text.

When compilation ends, intermediate text has been processed and converted to appropriate IRP (intermediate representation of a program). IRP, which is output from step 2 in Figure A-1 on page A-2, can be dynamically listed with the CODELIST parameter of the CL command CRTRPGPGM or can be listed at the end of compilation with an *LIST value for the GENOPT parameter on the CL command CRTRPGPGM. Refer to "Compiler Debugging Options" on page A-5 for explanations of these parameters and examples of IRP statements.

A program template is output from step 3 in Figure A-1 on page A-2. A template is the final form of a program before it is converted to an executable program, which is called an encapsulated program. A template can be listed at the end of a compilation with an *DUMP value for the GENOPT parameter on the CL command CRTRPGPGM. Refer to "Compiler Debugging Options" on page A-5 for explanation of this parameter and an example of a program template listing.

Compiler Phases

The compiler consists of the phases listed in Table A-1 on page A-3. These phases are shown in the order in which they are run.

If *NOGEN has been specified for the OPTION parameter on the CL command CRTRPGPGM, compilation ends following phase QRGCR.

During compilation, those phases that have a U in the third column in Table A-1 on page A-3 run unconditionally and those phases that have a C in the third column run only if they are required for the program being compiled.

The first compiler phase is named QRG1. All phases that follow QRG1 have names that begin with QRG and end with two identifying characters. These phases can be referred to by their identifying characters. For example, these characters can be used as values for debugging parameters in the CL command CRTRPGPGM. Refer to "Compiler Debugging Options" on page A-5 for more information.

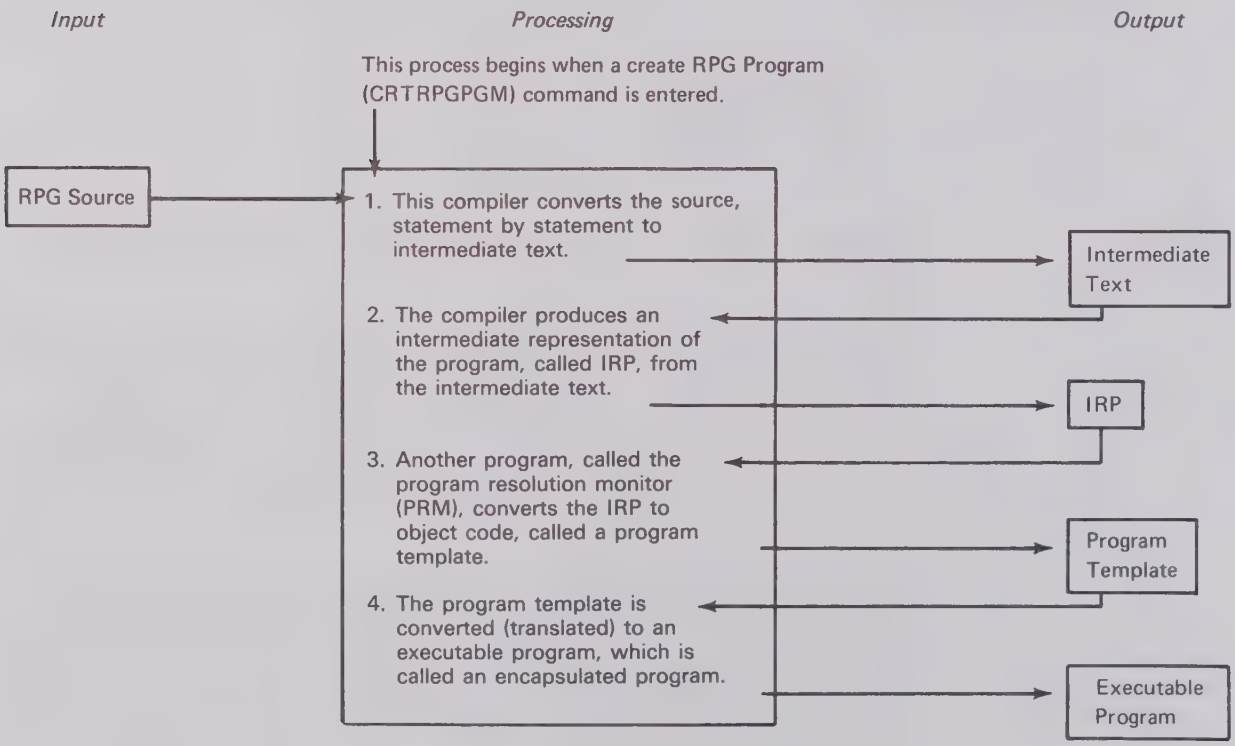


Figure A-1. Overview of the RPG Compiler

Table A-1 (Page 1 of 2). Compiler Phases

Phase Name	Phase Description	Called: Unconditionally (U) Conditionally (C)
QRG1	Command interface that receives control when the CRTRPGPGM command is entered, assigns defaults to the command parameter list, and passes the command parameter list to QRGRT.	U
QRGRT	Root phase that controls the calling of all other compiler phases and contains all system interfaces such as reading and printing records.	U
QRGSF	Phase that diagnoses the file description specifications and builds a file table.	C
QRGSE	Phase that diagnoses the extension specifications and builds intermediate text.	C
QRGSI	Phase that diagnoses the input specifications and builds intermediate text.	C
QRGSC	Phase that diagnoses the calculation specifications and builds intermediate text.	C
QRGSO	Phase that diagnoses the output specifications and builds intermediate text.	C
QRGAE	Phase that generates declare statements for fields and creates edit masks.	U
QRGDI	Phase that diagnoses relational errors among the source specifications. These errors are illogical or invalid combinations of entries.	U
QRGCR	Phase that produces a cross-reference listing, generates code for processing compile-time tables, and produces a list of compile-time messages upon completion of a compilation.	U
QRGGB	Phase that generates printer data.	U
QRGGC	Mainline phase that controls processing of arithmetic operations and other calculation operations.	C
QRGCC	Phase that is called by QRGGC to process compare calculation operations.	C
QRGYC	Phase that is called by QRGGC to process miscellaneous calculation operations.	C
QRGIC	Phase that is called by QRGGC to generate input/output linkages for calculation operations.	C
QRGMC	Phase that is called by QRGGC to generate code for MOVE calculation operations.	C

Table A-1 (Page 2 of 2). Compiler Phases

Phase Name	Phase Description	Called: Unconditionally (U) Conditionally (C)
QRGGI	Phase that generates code that extracts data from records and fills input fields.	C
QRGGO	Phase that generates code that builds output records.	C
QRGGS	Phase that generates data management for DISK, WORKSTN, and SEQ files.	C
QRGC1	Phase that generates code for getting input and processing multiple files.	U
QRGEC	Phase that generates subroutines required for the program, generates code for processing run-time tables, and generates beginning and ending code for the program.	U

Major Compiler Data Areas

The major compiler data areas are a common area (VCOMMON), a field name table (XFDTAB), a file name table (XFLTAB), a record name table (XRCTAB), and an indicator table (XINTAB).

Compiler Error Message Organization

Compiler error messages are organized according to the phases that issue them. For example, any compiler message beginning with 2 is issued by phase QRGSF. The following table lists compiler phases and the messages that they issue:

Error Messages	Phase
0000 to 1999	QRGRT
2000 to 2999	QRGSF
3000 to 3999	QRGSE
4000 to 4999	QRGSI
5000 to 5999	QRGSC
6000 to 6999	QRGSO
7000 to 7999	QRGDI and QRGCR
8000 to 8999	QRGAE and QRGCR

Run-Time Subroutines

Table A-2 on page A-5 lists the run-time subroutines that are used by the compiler.

Table A-2. Run-Time Subroutines

Subroutine Name	Subroutine Description	Called: Unconditionally (U) Conditionally (C)
QRGXINIT	Initializes the RPG/400 program.	U
QRGXDUMP	Provides a formatted dump of the RPG/400 program.	C
QRGXERR	Called when an error message is received.	C
QRGXPRT	Called by QRGXDUMP to print the dump.	C
QRGXSTAT	Called for the POST operation with a device specified in factor 1.	C
QRGXTIME	Called for the TIME operation code.	C
QRGXIOU	Called for the IN/OUT/UNLCK operation codes. Also used by the compiler to set the RETURNCODE data area and to retrieve the RPG/400 control specification data area during compilation.	U
QRGXMSG	Sends RPG/400 messages to the requester and provides the system dump when requested.	C
QRGXCLRF	Clears the file before a table dump at program end.	C
QRGXDSP	Called for the DSPLY operation code.	C
QRGXSIGE	Signals exception for run-time terminal error.	C
QRGXGDDM	Called for the CALL "GDDM" operation.	C
QRGINVX	Unlocks data areas when a program ends because of errors.	C
SUBR23R3	Message Retrieving	C
SUBR40R3	Moving Double-Byte data and deleting control characters.	C
SUBR41R3	Moving Double-Byte data and adding control characters.	C

Compiler Debugging Options

This section explains each of the debugging parameters. For examples of debugging information that can be requested by these parameters, refer to "Examples of Using Compiler Debugging Options" on page A-7.

*SOURCE Value for the OPTION Parameter

A value of *SOURCE for the OPTION parameter requests a listing of the RPG/400 source program. The default is *SOURCE.

*XREF Value for the OPTION Parameter

A value of *XREF for the OPTION parameter requests a cross-reference listing and a key field information table (when appropriate). Refer to Chapter 3, "Compiling an RPG/400 Program" for a description of this listing. The default is *XREF.

*DUMP Value for the OPTION Parameter

A value of *DUMP for the OPTION parameter causes the contents of major data areas such as VCOMMON, file name table, field name table, and IT (intermediate text) to be printed. This printing occurs only if the compiler ends abnormally. Therefore, *DUMP is usually specified when an unsuccessful compilation is retried. The default is *NODUMP.

*LIST Value for the GENOPT Parameter

A value of *LIST for the GENOPT parameter causes IRP, its associated hexadecimal code, and any error messages to be listed. The default is *NOLIST.

*ATR Value for the GENOPT Parameter

A value of *ATR for the GENOPT parameter causes the attributes for the IRP source to be listed. The listing includes the field descriptions and the statement numbers on which the fields are defined. The default is *NOATR.

*XREF Value for the GENOPT Parameter

A value of *XREF for the GENOPT parameter causes a cross-reference listing of all objects defined in the IRP to be printed when compilation ends.

*DUMP Value for the GENOPT Parameter

A value of *DUMP for the GENOPT parameter causes the program template to be listed. The default is *NODUMP.

*PATCH Value for the GENOPT Parameter

A value of *PATCH for the GENOPT parameter reserves space in the compiled program for a program patch area. The program patch area can be used for your debugging purposes. The size of the patch area is based on the size of the generated program. The default is *NOPATCH.

*OPTIMIZE Value for the GENOPT Parameter

A value of *OPTIMIZE for the GENOPT parameter causes the compiler to generate a program that runs more efficiently and that will possibly require less storage. However, specifying *OPTIMIZE can substantially increase the time required to create a program. The default is *NOOPTIMIZE. Existing programs may be optimized with the CL command CHGPGM command.

ITDUMP Parameter

For the CRTRPGPGM command, the ITDUMP parameter causes dynamic listing of intermediate text produced by a specified phase. Dynamic listing means that the intermediate text is printed during compilation while intermediate text is being built and stored. For the CRTRPTPGM command, the ITDUMP parameter causes a flow of the major routines run in one or more specified phases to be printed.

As many as 25 phases, each identified by the last two characters of its name, can be specified on the ITDUMP parameter. The list must be enclosed in parentheses. For example, the following ITDUMP parameter causes dynamic listing of intermediate text produced by QRGSE, QRGSO, and QRGSC: ITDUMP(SESOSC).

SNPDUMP Parameter

The SNPDUMP parameter produces a listing of major data areas and intermediate text following the running of one or more specified phases.

As many as 25 phases, each identified by the last two characters of its name, can be specified on the SNPDUMP parameter. The list must be enclosed in parentheses. For example, the following SNPDUMP parameter causes the listing of intermediate text produced by QRGSI, QRGSC, and QRGSO and also causes the contents of major data areas to be listed: SNPDUMP(SISCSO).

CODELIST Parameter

The CODELIST parameter causes dynamic listing of IRP produced by a specified phase. Dynamic listing means that the IRP is printed during compilation while the specified phase processes.

As many as 25 phases, each identified by the last two characters of its name, can be specified on the CODELIST parameter. The list must be enclosed in parentheses. For example, the following CODELIST parameter causes dynamic listing of IRP produced by QRGGC, QRGGO, and QRGEC: CODELIST(GCGOEC).

PHSTRC Parameter

The PHSTRC parameter specifies whether or not a phase trace occurs during compilation. A phase trace consists of the names of compiler phases being printed on the compiler listing in the order that the phases process. The numbers of the RXT messages (such as compiler headings) are also listed as they are retrieved.

The values that can be coded for the PHSTRC parameter are *YES and *NO. *NO is the default value.

Examples of Using Compiler Debugging Options

Figure A-2 on page A-9 shows examples of debugging information that can be requested by compiler debugging options on the CRTRPGPGM command. The compiler listing in Figure A-2 on page A-9 was printed for a CRTRPGPGM command that specified the following debugging parameters:

```
GENOPT(*LIST *DUMP) ITDUMP(SC) SNPDUMP(GO)
CODELIST(GO) PHSTRC(*YES)
```

The PHSTRC(*YES) parameter causes the name of a phase to be printed when the phase processes. For example, **A** in Figure A-2 shows that phase QRGSF processed the file description specification, phase QRGSE processed the extension specification, phase QRGSI processed the input specifications, and phase QRGSC processed the calculation specifications.

The ITDUMP(SC) parameter causes printing of intermediate text that phase QRGSC builds and stores. (See **B** in Figure A-2.)

The CODELIST(GO) parameter causes printing of IRP produced by phase QRGGO when that phase ends. (See **C** in Figure A-2.)

The SNPDUMP(GO) parameter causes printing of the contents of major data areas when phase QRGGO ends and causes printing of intermediate text produced by QRGGO. (See **D** in Figure A-2.)

The *LIST value for the GENOPT parameter causes printing of IRP and machine instructions when compilation ends. (See **E** in Figure A-2.) The headings in this IRP listing indicate the following information:

SEQ: A sequential numbering of the IRP statements. Error messages such as IRP syntax errors issued by the program resolution monitor use this number to refer to the IRP statements in error.

INST: A sequential numbering of the machine instructions generated from the IRP statements. Not all IRP statements cause machine instructions to be generated. The instruction number can be used as a breakpoint for OS/400 debugging functions. Refer to Chapter 4, "Error Messages, Testing and Debugging" or the *Programming: Control Language Programmer's Guide* for further information about breakpoints.

GENERATED CODE: Machine instructions that have been generated from IRP statements.

GENERATED OUTPUT: IRP statements.

BREAK: Breakpoints in the IRP that can be used for stopping points in OS/400 debugging functions. Refer to Chapter 4, "Error Messages, Testing and Debugging" or the *Programming: Control Language Programmer's Guide* for further information about breakpoints. If the breakpoint is a number, it indicates an RPG/400 source statement from which the IRP statement was generated.

The *DUMP value for the GENOPT parameter causes printing of the program template when compilation ends. (See **F** in Figure A-2.)

A value of *DUMP can be coded for the OPTION parameter to cause the contents of major compiler data areas to be printed if the compiler ends abnormally.

Figure A-3 on page A-24 shows an example of the information printed. For this example, the command is CRTRPGPGM OPTION(*DUMP).

```
RXT0001
RXT0002
RXT0003
RXT0004
RXT0028
 5728RG1 R01 M00 880715            IBM AS/400 RPG/400                  QGPL/DATAE          01/12/88  14:40:34      Page     1

RXT0005
 Compiler . . . . . . . . . . . . . :  IBM AS/400 RPG/400
RXT0020
 Command Options:
RXT0023
RXT0070
   Program  . . . . . . . . . . . . :  QGPL/DATAE
RXT0071
   Source file  . . . . . . . . . . :  *LIBL/QRPGSRC
RXT0072
   Source member  . . . . . . . . . :  *PGM
RXT0073
   Source listing options . . . . . :  *SOURCE     *XREF       *GEN        *NODUMP     *NOSECLVL
RXT0024
RXT0074
   Generation options . . . . . . . :  *LIST       *NOXREF     *NOATR      *DUMP       *NOOPTIMIZE
RXT0075
   Generation severity level  . . . :  9
RXT0076
   Print file . . . . . . . . . . . :  *LIBL/QSYSPRT
RXT0077
   Replace program  . . . . . . . . :  *YES
RXT0025
RXT0078
   User profile . . . . . . . . . . :  *USER
RXT0079
   Authority  . . . . . . . . . . . :  *CHANGE
RXT0080
   Text . . . . . . . . . . . . . . :  *SRCMBRTXT
RXT0081
   Phase trace  . . . . . . . . . . :  *YES
RXT0082
   Intermediate text dump . . . . . :  SC
RXT0083
   Snap dump  . . . . . . . . . . . :  GO
RXT0084
   Codelist . . . . . . . . . . . . :  GO
RXT0085
   Ignore decimal data error  . . . :  *NO
RXT0010
 Actual Program Source:
RXT0011
   Member . . . . . . . . . . . . . :  DATAE
RXT0012
   File . . . . . . . . . . . . . . :  QRPGSRC
RXT0013
   Library  . . . . . . . . . . . . :  QGPL
RXT0014
   Last Change  . . . . . . . . . . :  01/12/88  14:40:26
OPTIONS PASSED IN BY CL
```

Figure A-2 (Part 1 of 15). Examples of Compiler Debugging Information

```
5728RG1 R01 M00 880715           IBM AS/400 RPG/400                  QGPL/DATAE          01/12/88  14:40:34      Page     2

 SEQUENCE                                                                              IND   DO    LAST       PAGE   PROGRAM
 NUMBER    *...1....+....2....+....3....+....4....+....5....+....6....+....7...*  USE   NUM   UPDATE     LINE   ID
PHASE                        11010 11111010 00009DATAE     QGPL      QRPGSRC    LIBL                          QSYS
QRGRT         040000000000045000FFFFF4FFFFFFFF0FFFFFCCECC44444DCDD444444DDDCEDC4445DCCD444444444444444444444444DEEE
              0D0000000214C020110100111110100000094131500000877300000089772930O0C392300000000000000000000000008282
PHASE         PRT    LIBL                      SC                                                          GO
QRGRT         DDE4445DCCD444444444444444444444444EC444444444444444444444444444444444444444444444444444444CD44444
              793000C392300000000000000000000023000000000000000000000000000000000000000000000000000000007600000
PHASE                                                          GO
QRGRT         444444444444444444444444444444444444444444444444444CD444444444444444444444444444444444444444444
              000000000000000000000000000000000000000000000000000760000000000000000000000000000000000000000000
PHASE                      SRCMBRTXT                                                                 00 CHANGE   1
QRGRT         444444444444440005EDCDCDEEE444444444444444444444444444444444444444400000000000FF5CCCDCC444F00000000
              000000000000050AC2934293730000000000000000000000000000000000000000000000000C381575000100000000
PHASE                                                                        00000000
QRGRT         000000000000000000000000000000000000000000000000000000000000000000FFFFFFFF444444444444444444444444
              000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGRT         444444444444444444444440000000000000000000000000000000000000000000000000000000000000000000000000
              000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGRT         00000000000000000000000000000000
              00000000000000000000000000000000

5728RG1 R01 M00 880715           IBM AS/400 RPG/400                  QGPL/DATAE          01/12/88  14:40:34      Page     3

 SEQUENCE                                                                              IND   DO    LAST       PAGE   PROGRAM
 NUMBER    *...1....+....2....+....3....+....4....+....5....+....6....+....7...*  USE   NUM   UPDATE     LINE   ID
RXT0054
                    S o u r c e   L i s t i n g
     100  H                                                                                    12/09/87          A
                                                                                                    PHASE - SF            00027
     200  FQSYSPRT O   F     132          LPRINTER                                             01/12/88
                                                                                                    PHASE - SE            00000
     300  LQSYSPRT  30FL 250L                                                                  01/12/88
                                                                                                    PHASE - SI            00001
     400  IEXDS       UDS                                                                      01/12/88
     500  I                                    1   30F1                                        01/12/88
                                                                                                    PHASE - SC            00000
     600  C                    SETON                    LR                   1                 01/12/88
PHASE         C1  000006000                          LR
QRGSC         CF03FFFFFFFF0008808100062000000000000008DD000044444400    B
  00000600    3107000006000000000090008300000000000000390000000000
     700  C                    DO   100                                              B001   01/12/88
PHASE         C3  000007000                                                 100
QRGSC         CF05FFFFFFFF0008808100024000000000000000000000444444000000000FFF0000000000000000000000000000044
  00000700    330F000007000000800010005000000000000000000000000000000000000010000000000030300000000000000000
     800  C                    EXCPT                                 S           001   01/12/88
PHASE         C1  000008000
QRGSC         CF03FFFFFFFF0008808100033000000000000000000000444444400
  00000800    3107000008000000000090007B000000000000000000000000000000
     900  C                    END                                                   E001   01/12/88
PHASE         C1  000009000
QRGSC         CF03FFFFFFFF0008808100034000000000000000000000444444400
  00000900    3107000009000000000090005100000000000000000000000000000
    1000  C                    ADD  1       F1                                              01/12/88
PHASE         C3  000010000                                F1          1
QRGSC         CF05FFFFFFFF0008004000000000000000000000000CF4444000000000F0000000000000000000000000000044
  00001000    330F000010000000800100041000000000000000000006100000F000000001000000000001010000000000000000    A
                                                                                                    PHASE - SO            00000
    1100  OQSYSPRT EF 1                                                                         01/12/88
    1200  O                      PAGE                                                         01/12/88
RXT6103
              OVERFLOW INDICATOR OA ASSIGNED TO FILE QSYSPRT.
RXT0043
5728RG1 R01 M00 880715           IBM AS/400 RPG/400                  QGPL/DATAE          01/12/88  14:40:34      Page     4

            * * * * *  E N D   O F   S O U R C E  * * * * *
```

Figure A-2 (Part 2 of 15). Examples of Compiler Debugging Information

```
RXT0055
        A d d i t i o n a l   D i a g n o s t i c   M e s s a g e s
                                                                PHASE - D1            00000
RXT7999
RXT6999
                                                                PHASE - AE            00000
                                                                PHASE - CR            00000
 5728RG1 R01 M00 880715            IBM AS/400 RPG/400          QGPL/DATAE      01/12/88  14:40:34      Page     5

RXT9111
 TABLE OF END POSITION OFFSETS FOR FIELDS DESCRIBED USING POSITION NOTATION.
RXT9112
130
     1200      4
RXT9100
 5728RG1 R01 M00 880715            IBM AS/400 RPG/400          QGPL/DATAE      01/12/88  14:40:34      Page     6

RXT9104
                        C r o s s   R e f e r e n c e
RXT9126
 File and Record References:
RXT9108
       FILE/RCD    DEV/RCD     REFERENCES (D=DEFINED)
    01 QSYSPRT     PRINTER      200D  300  1100  1201
RXT9127
 Field References:
RXT9107
       FIELD       ATTR        REFERENCES (M=MODIFIED D=DEFINED)
* 7031 EXDS        DS(3)     400D
       F1          Z(3,0)    500D 1000M
       PAGE        P(4,0)   1200
       1           LITERAL  1000
       100         LITERAL   700
RXT9128
 Indicator References:
RXT9109
       INDICATOR  REFERENCES (M=MODIFIED D=DEFINED)
       LR          600M
       OA          200D 1201
RXT9122
     * * * * *  E N D   O F   C R O S S   R E F E R E N C E  * * * * *

RXT9132
 5728RG1 R01 M00 880715            IBM AS/400 RPG/400          QGPL/DATAE      01/12/88  14:40:34      Page     7

RXT9123
                        M e s s a g e   S u m m a r y
RXT9131
* QRG6103 Severity:  00   Number:    1
          Message . . . . :   No Overflow Indicator is specified but an
            indicator is assigned to a file and automatic skip to 6 is
            generated.
RXT9131
* QRG7031 Severity:  00   Number:    1
          Message . . . . :   The Name or indicator is not referenced.
RXT9124
     * * * * *  E N D   O F   M E S S A G E   S U M M A R Y  * * * * *
```

Figure A-2 (Part 3 of 15). Examples of Compiler Debugging Information

```
                                                                    PHASE - C1          00003
                                                                    PHASE - GC          00000
                                                                    PHASE - GO          00006 G
;/* PHASE - QRGGO     DATE - 11/05/87 *//*SVP*/                                              GO
                                                                                            GO
/************************************ GEN TIME -    0 ********//*ZCOMENT*/                   GO
                                                                                            GO
;ENTRY .OFL     INT/*ZLABX*/                                                                GO
                                                                                            GO
;DCL INSPTR .OFLWRTN/*OVERFLOW RETURN POINTER*//*ZDCLRTN*/                                   GO
                                                                                            GO
;BRK '1201    '/*BRK POINT*//*ZBRKPT*/                                                      GO
                                                                                            GO
;CMPBLA(B) *IN1A,*OFF/ EQ(.OOR0001)/*COND IND TST*//*ZCONID*/                               GO
                                                                                            GO
;SPACE 3                                                                                    GO
;/*     QSYSPRT   FILE OUTPUT                                    *//*ZFLCOM*/             GO
                                                                                            GO
;SETSPP .FIBPTR,.F01FIB/*SET FIB PTR*//*ZFIBPT*/                                            GO
                                                                                            GO
;CPYBLA .CURROP, C'WRITE'/* SET OP *//*ZCURROP*/                                            GO
                                                                                            GO
;CPYBLA .PRTCTL,X'0000000000060000'/*SET OPERATION*//*ZPRC*/                                GO
                                                                                            GO
;CPYBWP .BUFPTR,.U01BUF0/*LOCATE BUFFER*//*ZBUFADD*/                                        GO
                                                                                            GO
;CPYBLA  .BUFFER(1:0132),.BLANKS      /*ZCLRBUF*/                                         GO
                                                                                            GO
;CPYBLA .F01XSET,'00000WRITEF*OFL    1201           '/*FILL FEEDBACK*/                     GO
;CPYBLA .F01EIND,*OFF/*NO ERROR IND*//*ZFIBPT0*/                                            GO
                                                                                            GO
;CMPBLA(B) .F010PEN,*OFF/EQ(.DMEXL2)/*FILE OPEN?*//*ZFIBPT1*/                              GO
                                                                                            GO
;CALLI .XRVFW01,*,.DRIVRTN/*ZPUT*/                                                          GO
                                                                                            GO
;.OOR0001:/*LABEL*//*ZLAB*/                                                                 GO
                                                                                            GO
;CPYBLA .OFL2A(2:2),'00'/*SET INTERNAL IN D OFF*//*ZOF2OFF*/                                GO
                                                                                            GO
;B .OFLWRTN/*RETURN*//*ZOFLRTN*/                                                            GO
```

Figure A-2 (Part 4 of 15). Examples of Compiler Debugging Information

```
                                                                                                                    GO
 5728RG1 R01 M00 880715            IBM AS/400 RPG/400                  QGPL/DATAE          01/12/88  14:40:34      Page     8

;ENTRY .EXCPT   INT/*ZLABX*/                                                                                        GO
                                                                                                                    GO
;DCL INSPTR .LINERTN/*EXCPTION RETURN POINTER*//*ZDCLLRT*/                                                          GO
                                                                                                                    GO
;BRK '1100    '/*BRK POINT*//*ZBRKPT*/                                                                            GO
                                                                                                                    GO
;SPACE 3                                                                                                            GO
;/*     QSYSPRT   FILE OUTPUT                                    *//*ZFLCOM*/                                     GO
                                                                                                                    GO
;SETSPP .FIBPTR,.F01FIB/*SET FIB PTR*//*ZFIBPT*/                                                                    GO
                                                                                                                    GO
;CPYBLA .CURROP, C'WRITE'/* SET OP *//*ZCURROP*/                                                                    GO
                                                                                                                    GO
;CMPBLA(B) *IN1A,*ON/NEQ(.OEC0001)/*OVERFLOW ?*/                                                                    GO
;CALLI .FETCH01,*,.OFETCHR/*FETCH OVERFLOW*/                                                                        GO
;.OEC0001:/*RET/BR TGT*//*ZPRTOF*/                                                                                  GO
                                                                                                                    GO
;CPYBLA .PRTCTL,X'0000000100000000'/*SET OPERATION*//*ZPRC*/                                                        GO
                                                                                                                    GO
;CPYBWP .BUFPTR,.U01BUFO/*LOCATE BUFFER*//*ZBUFADD*/                                                                GO
                                                                                                                    GO
;CPYBLA  .BUFFER(1:0132),.BLANKS      /*ZCLRBUF*/                                                                   GO
                                                                                                                    GO
;BRK '1200    '/*BRK POINT*//*ZBRKPT*/                                                                            GO
                                                                                                                    GO
;ADDN(S) PAGE    ,1   /*INCREMENT*//*ZINC*/                                                                        GO
                                                                                                                    GO
;CVTNC .BUFFER(0001:0004),  PAGE              ,X'02000400000000'/*NUM TO CHAR*//                                    GO
*ZNNUM*/                                                                                                            GO
                                                                                                                    GO
;OR(S) .BUFFER(0004:1),X'F0'/*FORCE POS*/                                                                           GO
;TSTRPLC .BUFFER(0001:0004),' '/*ZERO SUPPRESS*//*ZZEDT*/                                                           GO
                                                                                                                    GO
;CPYBLA .F01XSET,'00000WRITEF*EXCP   1100            '/*FILL FEEDBACK*/                                            GO
;CPYBLA .F01EIND,*OFF/*NO ERROR IND*//*ZFIBPT0*/                                                                    GO
                                                                                                                    GO
;CMPBLA(B) .F01OPEN,*OFF/EQ(.DMEXL2)/*FILE OPEN?*//*ZFIBPT1*/                                                       GO
                                                                                                                    GO
;CALLI .XRVFW01,*,.DRIVRTN/*ZPUT*/                                                                                  GO
                                                                                                                    GO
;B .LINERTN/*BRANCH*//*ZUNBR*/                                                                                      GO
                                                                                                                    GO
;DCL INSPTR .OFETCHR/*OVERFLOW FETCH RETURN*//*ZFETDCL1*/                                                           GO
                                                                                                                    GO
;DCL DD .SAVOFOA CHAR(3)/*OVFL IND SAVE*//*ZFETDCL2*/                                                               GO
                                                                                                                    GO
;ENTRY .FETCH01 INT/*OVRFLW FETCH SUBR*//*ZFETCH1*/                                                                 GO
                                                                                                                    GO
;CALLI .OFL,*,.OFLWRTN/*GOTO OVERFLOW ROUTINE*//*ZFETCH3*/                                                          GO
                                                                                                                    GO
;B .OFETCHR/*RETURN*//*ZFETCH5*/                                                                                    GO
```

Figure A-2 (Part 5 of 15). Examples of Compiler Debugging Information

```
                                                                                                                                GO
VCOMMON AREA
 5728RG1 R01 M00 880715            IBM AS/400 RPG/400                  QGPL/DATAE          01/12/88  14:40:34      Page     9

PHASE                      H          D                        QRGGO   11 05 87                  PAGE      PAGE
QRGGO         0002000600004000C0000000A000C0304000000000000000000000DDCCD444FF6FF6FF44000100000000000DCCC444444DCCC      D
              0000000500030018010000F010004000005000200000003000C008977600011105187000700F0000000600007175000000071750
PHASE                  10 2QRPGSRC       QGPL
QRGGO         44444444FF4FDDDCEDC4444444DCDD4444444444E000000000000000000000000000000000000000000000000000000000000000
              0000000010028977293000000087730000000000F000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
              0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000780
              0000000000000020000000000000000000000020000000000000000000000000000000000000000000000000000000000000000181
PHASE                           0                                                                  1000000000000000000000000000
QRGGO         0446446E000400000000D8000000000000000000000000000000000000004448FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
              01010BB000030000000006000000000000000000000000000000000000000000000000000000010000000000000000000000000000
PHASE         0000100000000000000000000000010000110000000000000000000000000000000000000000000000000000000000000000
QRGGO         FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF4000000000
              0000100000000000000000000000010000110000000000000000000000000000000000000000000000000000000000000000000000
PHASE               703100   1
QRGGO         00000FFFFFF444F444444444444444444444444444444444444444444444444444444444444444444444444444444444444444
              0000070310000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE                          1440340112880001000012010000 1201DATAE
QRGGO         444444444440000000005FFFFFFFFFFFFFFFFFFFFFFFFFFFFFCCECC4444400000000000000000000000000000000000000000000
              0000000000000000000B1440340112880001000012010000120141315000000900020000000000000000000000000000000000
PHASE
QRGGO         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
              0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
              0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
              0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
              0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
              0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
              0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure A-2 (Part 6 of 15). Examples of Compiler Debugging Information

```
5728RG1 R01 M00 880715          IBM AS/400 RPG/400              QGPL/DATAE          01/12/88  14:40:34      Page     10

PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure A-2 (Part 7 of 15). Examples of Compiler Debugging Information

```
5728RG1 R01 M00 880715          IBM AS/400 RPG/400              QGPL/DATAE          01/12/88  14:40:34      Page     11

PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO    0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
         0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure A-2 (Part 8 of 15). Examples of Compiler Debugging Information

```
5728RG1 R01 M00 880715          IBM AS/400 RPG/400              QGPL/DATAE        01/12/88  14:40:34     Page    12

PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE
QRGGO       00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
PHASE         00          RPG
QRGGO       00FF000000000000DDC4444444800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000009770000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
FILENAME TABLE
PHASE       QSYSPRT                              00000200             1           0A   0
QRGGO       DEEEDDE400000000000000000000000000FFFFFFFF08010100000000F000000000000DC444D44240080000000000000000040
            8282793000000000000000000000000000000020004090E01000000100000000000061000600040010000000000000000000
PHASE
QRGGO       00000000000000000000444444444444000000000000000000000000000000000000000000000000000000000000000000000000
            00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
RECORD NAME TABLE
FIELD NAME TABLE
PHASE         IN
QRGGO       5CD4440000040806000000000000000000000000000000000000000000000000000000
            C9500001001000030010000000000000000000000000000000000000000000000000000
PHASE       UDATE     B
QRGGO       ECCEC400000C000000000000000000000000000000000000000000000000000000000000
            4413500600320100001000000000000000000000000000000000000000000000000000
PHASE       UDAY      B
QRGGO       ECCE4400000C000000000000000000000000000000000000000000000000000000000000
            4418000200320100001000000000000000000000000000000000000000000000000000
```

Figure A-2 (Part 9 of 15). Examples of Compiler Debugging Information

```
5728RG1 R01 M00 880715            IBM AS/400 RPG/400                  QGPL/DATAE         01/12/88  14:40:34      Page     13

PHASE         UMONTH     B
QRGGO         EDDDEC00000C0000000000000000000000000000000000000000000000000000000000000000
              4465380200320100001000000000000000000000000000000000000000000000000000000000
PHASE         UYEAR      B
QRGGO         EECCD400000C0000000000000000000000000000000000000000000000000000000000000000
              4851900200320100001000000000000000000000000000000000000000000000000000000000
PHASE         PAGE       0
QRGGO         DCCC4400000F0000000000000000000000000000000000000000000000000000000000000000
              7175000400300800001100800000000000000000000000000000000000000000000000000000
PHASE         PAGE1
QRGGO         DCCCF400000C0000000000000000000000000000000000000000000000000000000000000000
              7175100400300800001000000000000000000000000000000000000000000000000000000000
PHASE         PAGE2
QRGGO         DCCCF400000C0000000000000000000000000000000000000000000000000000000000000000
              7175200400300800001000000000000000000000000000000000000000000000000000000000
PHASE         PAGE3
QRGGO         DCCCF400000C0000000000000000000000000000000000000000000000000000000000000000
              7175300400300800001000000000000000000000000000000000000000000000000000000000
PHASE         PAGE4
QRGGO         DCCCF400000C0000000000000000000000000000000000000000000000000000000000000000
              7175400400300800001000000000000000000000000000000000000000000000000000000000
PHASE         PAGE5
QRGGO         DCCCF400000C0000000000000000000000000000000000000000000000000000000000000000
              7175500400300800001000000000000000000000000000000000000000000000000000000000
PHASE         PAGE6
QRGGO         DCCCF400000C0000000000000000000000000000000000000000000000000000000000000000
              7175600400300800001000000000000000000000000000000000000000000000000000000000
PHASE         PAGE7
QRGGO         DCCCF400000C0000000000000000000000000000000000000000000000000000000000000000
              7175700400300800001000000000000000000000000000000000000000000000000000000000
PHASE         EXDS
QRGGO         CECE4400000500000002000000000000000000000000000000000000000000000000000000000
              5742000300110000002001000000000000000000000000000000000000000000000000000000
PHASE         F1         1
QRGGO         CF444400000F2000000000100000000000000000000000000000000000000000000000000000
              6100000300210000010101000000000000000000000000000000000000000000000000000000
INDICATOR TABLE
PHASE         1P   0A   0B   0C   0D   0E   0F   0G   0V   MR   L1   L2   L3   L4   L5   L6   L7   L8   L9   LR
QRGGO         FD000DCC40DC040DC040DC040DC040DC040DC040DC040DE040DD002DF02BDF02BDF02BDF02BDF02BDF02BDF02BDF02BDDC09
              172026100F6200F6300F6400F6500F6600F6700F6500F4910A3100E3200E3300E3400E3500E3600E3700E3800E3900E3902E
```

Figure A-2 (Part 10 of 15). Examples of Compiler Debugging Information

```
5728RG1 R01 M00 880715          IBM AS/400 RPG/400                QGPL/DATAE        01/12/88  14:40:34      Page     14

PHASE         H1   H2   H3   H4   H5   H6   H7   H8   H9   KA   KB   KC   KD   KE   KF   KG   KH   KI   KJ   KK
QRGGO         CF01ECF01ECF01ECF01ECF01ECF01ECF01ECF01ECF01EDC000DC000DC000DC000DC000DC000DC000DC000DC000DD000DD000
              8100E8200E8300E8400E8500E8600E8700E8800E8900E2104E2204E2304E2404E2504E2604E2704E2804E2904E2104E2204E
PHASE         KL   KM   KN   KP   KQ   KR   KS   KT   KU   KV   KW   KX   KY   RT   U1   U2   U3   U4   U5   U6
QRGGO         DD000DD000DD000DD000DD000DD000DE000DE000DE000DE000DE000DE000DE00EEF08EEF08EEF08EEF08EEF08EEF08E
              2304E2404E2504E2704E2804E2904E2204E2304E2404E2504E2604E2704E2804E9308E4100E4200E4300E4400E4500E4600E
PHASE         U7   U8   01   02   03   04   05   06   07   08   09   10   11   12   13   14   15   16   17   18
QRGGO         EF08EEF08EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00E
              4700E4800E0101F0201F0301F0401F0501F0601F0701F0801F0901F1001F1101F1201F1301F1401F1501F1601F1701F1801F
PHASE         19   20   21   22   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38
QRGGO         FF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00E
              1901F2001F2101F2201F2301F2401F2501F2601F2701F2801F2901F3001F3101F3201F3301F3401F3501F3601F3701F3801F
PHASE         39   40   41   42   43   44   45   46   47   48   49   50   51   52   53   54   55   56   57   58
QRGGO         FF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00E
              3901F4001F4101F4201F4301F4401F4501F4601F4701F4801F4901F5001F5101F5201F5301F5401F5501F5601F5701F5801F
PHASE         59   60   61   62   63   64   65   66   67   68   69   70   71   72   73   74   75   76   77   78
QRGGO         FF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00E
              5901F6001F6101F6201F6301F6401F6501F6601F6701F6801F6901F7001F7101F7201F7301F7401F7501F7601F7701F7801F
PHASE         79   80   81   82   83   84   85   86   87   88   89   90   91   92   93   94   95   96   97   98
QRGGO         FF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00EFF00E
              7901F8001F8101F8201F8301F8401F8501F8601F8701F8801F8901F9001F9101F9201F9301F9401F9501F9601F9701F9801F
PHASE         99
QRGGO         FF00E
              9901F
INTERMEDIATE TEXT I.T.
PHASE         RL  000003000     QSYSPRT
QRGGO         DD02FFFFFFFFF00000DEEEDDE4000000008
  00000300    9303000003000001008282793000000000
PHASE         ID  000004000
QRGGO         CC01FFFFFFFFF0000
  00000400    940100000400000E0
PHASE         ID  000005000
QRGGO         CC01FFFFFFFFF0000
  00000500    940100000500000F2
PHASE         C1  000006000                          LR
QRGGO         CF03FFFFFFFFF0008808100062000000000000008DD000044444400
  00000600    3107000006000000000090008300000000000000039000000000000
PHASE         C3  000007000                                                  100
QRGGO         CF05FFFFFFFFF0008008100024000000000000000000000444444000000000FFF000000000000000000000000000044
  00000700    330F000007000000080010005000000000000000000000000000000000000100000000000303000000000000000

5728RG1 R01 M00 880715          IBM AS/400 RPG/400                QGPL/DATAE        01/12/88  14:40:34      Page     15

PHASE         C1  000008000
QRGGO         CF03FFFFFFFFF0008808100033000000000000000000044444400
  00000800    3107000008000000000090007B000000000000000000000000000
PHASE         C1  000009000
QRGGO         CF03FFFFFFFFF0008808100034000000000000000000044444400
  00000900    3107000009000000000090005100000000000000000000000000
PHASE         C3  000010000                                  F1            1
QRGGO         CF05FFFFFFFFF0008004000000000000000000000000CF444400000000F000000000000000000000000044
  00001000    330F000010000000080010004100000000000000000006100000F000000001000000000001010000000000000
PHASE         OR  000011000       QSYSPRT                  1
QRGGO         DD03FFFFFFFFF0000008DEEEDDE444444420044444444404F4444
  00001100    690500001100000100048282793000000080000000000010000
PHASE         OF  000012000                      PAGE          Z
QRGGO         DC05FFFFFFFFF000000000000000000000DCCC440000000060E444444444000000000000000000000000000000000
  00001200    660E000012000004040000000000000060071750000000053009000000000000000000000000000000000000000000
PHASE         OR  000012010       QSYSPRT          0A        06
QRGGO         DD03FFFFFFFFF0000008DEEEDDE40000009004DC444444044FF44
  00001201    690500001201000100048282793000000000061000000000600

                                                                                PHASE - GI          00016
                                                                                PHASE - GB          00000
                                                                                PHASE - GS          00005
                                                                                PHASE - EC          00001
```

Figure A-2 (Part 11 of 15). Examples of Compiler Debugging Information

```
 5728RG1 R01 M00 880715            IBM AS/400 RPG/400                 QGPL/DATAE         01/12/88  14:40:34      Page     16

RXT9125
                          F i n a l   S u m m a r y
RXT0045
 Message Count:  (by Severity Number)
RXT0046
           TOTAL    00     10     20     30     40     50
RXT0047
             2       2      0      0      0      0      0
RXT0060
 Program Source Totals:
RXT0061
    Records . . . . . . . . . . :   12
RXT0062
    Specifications  . . . . . . :   12
RXT0063
    Table Records . . . . . . . :   0
RXT0064
    Comments  . . . . . . . . . :   0
 COUNT OF SERVICE REQUESTS
 GET SOURCE                000011
 PRINT SOURCE              000011
 PRINT LINE                000032
 RELATIONAL DIAGNOSTIC     000001
 SYNTAX DIAGNOSTIC         000000
 PRINT ERR NUMBER          000001
 GET I.T.                  000044
 UPDATE I.T.               000014
 BUILD I.T.                000011
 NOTE I.T.                 000005
 POINT I.T.                000011
 PUT IRP STRING            000192
 INVALID REQUEST           000000
 VERIFY NAME SYNTAX        000000
 CANCEL                    000000
 TRACE                     000000
 PRINT LINE & DIAG         000001
 INVALID REQUEST           000004
 INVALID REQUEST           000017
 INVALID REQUEST           000014
 TOTAL SERVICE REQUESTS -  000369
```

Figure A-2 (Part 12 of 15). Examples of Compiler Debugging Information

```
TEMPLATE PASSED TO PRM
PHASE
QRGEC         8000000000A6800080000000000C040538000000000C040488000000000C0404A8000000000C040408000000000A680000000
              000000000540E010000000000500C060000000000500C0C000000000500C014000000000500C004000000000540C0100000
PHASE                                                                                                    DATAE
QRGEC         0000000000008000000000C040610000000000000000000000000000000000000000000000000000000000CCECC44444E000F2800000
              0000000000000000000000500C04D000000000000000000000000000000000000000000000000000000041315000000000C0000000

 5728RG1 R01 M00 880715             IBM AS/400 RPG/400                    Q6PL/DATAE          01/12/88  14:40:34      Page     17

PHASE                 QGPL
QRGEC         00000000DCDDE4444400000000000000000000B80000D500000000000000000000000000000000000000000000000000000000
              0000100087730000000000000000000000000A00000C208000000000000000000000000000000000000000000000000000000
PROGRAM DATA AREA
PHASE         5728RG1R01 M0YQRPGSRC   DATAE     QGPL
QRGEC         FFFFDCFDFF4DFEDDDCEDC444CCECC44444DCDD4444444444444444444
              5728971901040889772930004131500000877300000000000000000000
AUTHORIZATION PARM
PHASE            CHANGE
QRGEC         005CCCDCC444
              01C381575000
USER TEXT
PHASE
QRGEC         0044444444444444444444444444444444444444444444444444444
              0000000000000000000000000000000000000000000000000000000
WHERE USED DATA
PHASE                EXDS        LIBL        QSYSPRT     LIBL        QSYSPRT
QRGEC         000400CECE44444445DCCD44444403DEEEDDE44445DCCD44444410DEEEDDE4444000
              0004025742000000C392300000B08282793000C392300000091828279300002000
```

Figure A-2 (Part 13 of 15). Examples of Compiler Debugging Information

```
RXT0034
 PRM has been called.                                                                                                E
 5714SS1 01/12/88 14:40:34                          GENERATED OUTPUT                                  01/12/88 14:40:34  Pag  18
  SEQ   INST          GENERATED CODE     *... ... 1 ... ... 2 ... ... 3 ... ... 4 .. . ... 5 ... ... 6 ... ... 7 ... ... 8  BREAK
00001                                                                                                                ;
00002                                      BRK '.ENTRY  '                          /*Z1STBRK*/                    ;  .ENTRY
                                                                                                                        .ENTRY
                                           /* PHASE - QRGSF     DATE - 06/26/87 */                                    .ENTRY
                                           /*SVP*/                                                                     .ENTRY
00003                                                                                                                ;  .ENTRY
                                           /* PHASE - QRGSE     DATE - 02/12/82 */                                    .ENTRY
                                           /*SVP*/                                                                     .ENTRY
00004                                                                                                                ;  .ENTRY
                                           /* PHASE - QRGSI     DATE - 05/15/86 */                                    .ENTRY
                                           /*SVP*/                                                                     .ENTRY
00005                                                                                                                ;  .ENTRY
                                           /* PHASE - QRGSC     DATE - 05/04/87 */                                    .ENTRY
                                           /*SVP*/                                                                     .ENTRY
00006                                                                                                                ;  .ENTRY
                                           /* PHASE - QRGSO     DATE - 06/27/86 */                                    .ENTRY
                                           /*SVP*/                                                                     .ENTRY
00007                                                                                                                ;  .ENTRY
                                           /* PHASE - QRGD1     DATE - 11/26/87 */                                    .ENTRY
                                           /*SVP*/                                                                     .ENTRY
00008                                                                                                                ;  .ENTRY
                                           /* PHASE - QRGAE     DATE - 11/26/87 */                                    .ENTRY
                                           /*SVP*/                                                                     .ENTRY
00009                                                                                                                ;  .ENTRY
00010  0001 000004  0252 0009 0257        SETIEXIT .RPGXIEX,.RPXIEXP              /*SET UP INVOCATION EXIT P      .ENTRY
                                                                                   GM*/                          ;  .ENTRY
00011  0002 00000A  1011 0234             B .START                                /*BEGIN OF PROGRAM */           ;  .ENTRY
00012                               .STOP:                                                                           ;  .ENTRY
00013  0003 00000E  30B2 0007 000D        CPYBLA .INVOCSW,*OFF                    /*ALLOW CALL TO THIS PGM*/      .ENTRY
00014  0004 000014  02A1 0000             RTX *                                   /* RETURN */                      .ENTRY
                                                                                   /*ZSTAR*/                       ;  .ENTRY
                                                                                                                        .ENTRY
                                           /*START OF THE PROGRAM*/                                                    .ENTRY
00015                                                                                                                ;  .ENTRY
                                           /* STATIC AREA FOR INDIC/FIELDS */                                           .ENTRY
00016                                                                                                                ;  .ENTRY
                                           /* START OF STRUCTURE POINTED BY .DMPTRLT*/                                  .ENTRY
00017                                                                                                                ;  .ENTRY
00018                                      DCL DD .RPGPGM CHAR(1) BDRY(16) INIT                                      ;  .ENTRY
00019                                      DCL DD .FIRSTSW CHAR(1) INIT('0')       /* PROGRAM CALLED BEFORE *       .ENTRY
                                                                                   /                             ;  .ENTRY
00020                                      DCL DD .EOJSW CHAR(1) INIT('0')         /*PROGRAM WENT TO EOJ*/       ;  .ENTRY
00021                                      DCL DD .DUMPSW CHAR(1) INIT('0')        /* DUMP REQUESTED*/            ;  .ENTRY
00022                                      DCL DD .ERRTERM CHAR(1) INIT('0')       /*PROGRAM TERMINATED */       ;  .ENTRY
00023                                      DCL DD .INVOCSW CHAR(1) INIT('0')                                         ;  .ENTRY
00024                                      DCL DD .INVOCER CHAR(4) INIT('8888')                                      ;  .ENTRY
00025                                      DCL SYSPTR .RPGXIEX INIT('QRGXINVX', TYPE(PGM,1))                            .ENTRY
                                                                                   /*INVOCATION EXIT PROGRAM*      .ENTRY
                                                                                   /                             ;  .ENTRY
00026                                      DCL DD .INVXLVL BIN(2) INIT(1)          /*INTERFACE LEVEL FOR QRGX      .ENTRY
                                                                                   INVX*/                           .ENTRY
                                                                                   /*END OF STATIC AREA STRUC      .ENTRY
                                                                                   TURE*/                        ;  .ENTRY
00027                                      DCL DD .BLANKS CHAR(140) INIT((140)' ')                                   ;  .ENTRY
00028                                      DCL CON *ON CHAR(1) INIT('1')           /* SET/CHECK INDICATORS ON       .ENTRY
```

Figure A-2 (Part 14 of 15). Examples of Compiler Debugging Information

```
                                                                    */                  ; .ENTRY
5714SS1 01/12/88 14:40:34                    GENERATED OUTPUT                  01/12/88 14:40:34  Pag  56
  MSGID                          MI INSTRUCTION STREAM SEMANTICS DIAGNOSTICS
5714SS1 01/12/88 14:40:34                    GENERATED OUTPUT                  01/12/88 14:40:34  Pag  57
 OFFSET                                                          I TEMPLATE DISPLA
 00000000      00003272  00000000  0201C4C1  E3C1C540  40404040  40404040  40404040  40404040      F
 00000020      40404040  40404040  C0000000  00000000  00000000  00010000  00000000  00000000
 00000040      00000000  00000000  000502FD  DD000400  00000000  00000000  00000000  00000000
 00000060      208000FC  00000000  00000000  019602B1  00000100  00000C14  0000171C  00000008
 00000080      0000015E  00003114  00000000  00000A4B  000026C8  00000000  00000196  00000000
 000000A0      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
 000000C0      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
 000000E0      00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000
 00000100      00000AD2  02520009  02571011  023430B2  0007000D  02A10000  21320050  020D10B2
 00000120      021F000C  1042020B  20021022  022F0059  0283004C  022D0000  1C464000  021623CA
 00000140      026A1C46  C0000216  20000059  10B2021F  000D1011  005C30B2  021F000D  10B2002E
 00000160      002D10B2  002D005A  10B20231  20D710B2  0232002D  1CC24000  0005000C  02691CC2
 00000180      4000005B  000C0269  10110061  30B20030  005E10B2  00120014  11930013  00141CC2
 000001A0      40000016  000D005F  10BE0015  000C1011  006430BE  0015000D  1CC24000  0060000C
 000001C0      006410B2  0060000C  10110068  30B20231  20D710B2  0232002D  10B2003A  00AB10B2
 000001E0      003B00E9  1CC24000  005B000C  006210B2  005B000C  10110269  30B20232  00631011
 00000200      026930B2  00300065  30B20030  006730B2  00300069  1CC24000  0016000C  02660293
 00000220      00740000  006A30B2  0030006C  10B20016  000C1042  006D2001  3C461000  006D006F
 00000240      00700293  007A0000  007B3143  006D2001  1011006E  3143001B  00711011  007230B2
 00000260      00300073  1011005D  3CC24000  0013000D  00780082  00840154  10B20022  007510B2
 00000280      01C80076  01320087  01A010B2  608F2001  2084000B  10B20190  00771082  016C000D
 000002A0      1CC24000  0174000D  01FF0293  01ED0000  009030B2  60112002  20020079  1011006A
 000002C0      20820084  015410B2  0022007C  1CC2C000  0013000C  007D0293  00830000  008130B2
 000002E0      01C8007E  01320087  01A010B2  608F2001  2084000B  11430019  200110A3  608F2001
 00000300      20040019  007F1197  608F2004  200120F0  10A2608F  20012004  204010B2  01900080
 00000320      10B2016C  000D1CC2  40000174  000D01FF  029301ED  00000090  3011007B  22930074
 00000340      0000006A  30110081  21320085  00AC0132  0086011B  1042008E  409D2001  013201CD
 00000360      4089008E  013201CF  00AE0132  01CE00AF  104201D4  01C91042  01D601CA  104201D3
 00000380      01CB1042  01D501CC  10B20220  000C0283  01CD01D0  000010B2  0220000D  10B20141
 000003A0      01420083  0088011F  014410B2  01100135  1CC24000  00C4000C  01D10132  01D900B2
 000003C0      10B201DB  000C3042  01D701C9  104201D8  01CB0293  01E00000  01C73042  01D701CA
 000003E0      104201D8  01CC0293  01E00000  01C73011  01D13C46  400001D8  200001E7  1C462000
 00000400      01D800B8  01E41C46  100001D8  00B801E1  1C461000  00B800B9  01E31011  01E43C46
 00000420      900001D8  00B901E3  1C461000  00B800B9  01E310B2  81DA01D9  01E23042  00B801D8
 00000440      101101E7  3CC24000  00DE01E5  01E310B2  81DA01D9  01E61011  01E33C46  400001D7
 00000460      200001C7  1C461000  00B800B9  01EA1143  00B801D7  1C469000  00B800B9  01C710B2
 00000480      81DA01D9  01E81011  01C73011  01C73143  00B801D7  1C469000  00B800BA  01C71147
 000004A0      00B800BA  101101C7  3CC24000  0174000D  01FF0293  01DD0000  01D13011  009030B2
 000004C0      0220000D  10B200E1  022110B2  00E301F1  1CC24000  01EF000C  01FB1CC2  400000DD
 000004E0      01F201FD  1CC24000  00DD01F3  01FD0132  008500AC  1CE2C000  011B0000  01F410B2
 00000500      00CB000D  10B200C1  000D10B2  00C2000C  10B200DB  000C10B2  00D2000C  10B200DA
 00000520      000D1011  01F730B2  01410142  00830088  011F0105  10B20110  01351011  01F730B2
 00000540      00DC01F6  30B20225  000D10B2  0220000D  10B2003A  00AB10B2  003B00E9  10B20231
 00000560      20C610B2  023300DC  01320213  00841CC2  400000C3  000C0090  10114202  00B33CC2
 00000580      200000DC  01F90268  1CC21000  00DC01FA  02681011  009030B2  00DC01FC  10B20231
 000005A0      20C610B2  023300DC  01320213  008410B2  003A00AB  10B2003B  00E91011  01F830B2
 000005C0      00DC01FE  101101F7  30B200DC  02001011  01F730B2  01410142  00830088  011F0105
 000005E0      10B20110  01351011  01F73CC2  C0000007  000C0235  1042022B  200210B2  02400008
 00000600      1011026B  30B20007  000C10B2  00300236  10B2002F  02371042  022B2000  1CC24000
 00000620      0003000C  023C0547  020E0000  20010132  00270000  1042020B  20020283  0230022C
 00000640      00001C46  40000216  23CA0282  03EF0298  029C2001  10B20028  003F0082  02520245
 00000660      1042020B  20010283  02550254  000010B2  0232002D  1C46C000  02162000  026A10B2
 00000680      01C5002C  10B201C6  00361042  00232001  10220090  023A2132  00844204  002310B2
 000006A0      00E90239  10B200C3  000D1011  025E3D47  C0000023  20010238  10B2021E  000B10B2
 000006C0      002F023B  10B201EF  000D1042  022B2000  23EF0298  029C2001  1CC24000  0003000C
 000006E0      005D10B2  0003000C  10110072  3042020B  20020283  023D0243  00001C46  40000216
 Program DATAE is placed in library QGPL. 00 highest Error-Severity-Code.
RXT0048
        ***** END OF COMPILATION *****
```

Figure A-2 (Part 15 of 15). Examples of Compiler Debugging Information

```
EXCEPTION DATA
PHASE                CPFCPF2410RPGRT
RPGEC      0009000999010CDCDCFFFFDDCDF4444401DJ2J4444443D3DDDJDDD
```

IRP Layout

The RPG/400 compiler generates code that runs in a fixed program cycle, which is described in "RPG/400 Program Cycle" on page 1-5. A programmer can modify this cycle to some extent with branching operations, processing subroutines, and exception/error handling subroutines. Because the cycle is similar for all user programs, common segments of code appear in the IRP generated for most programs. Knowing what these segments do and where they can be found in an IRP listing can help you relate a user program to the IRP that the compiler generates.

Figure A-4 on page A-26 identifies labels that appear in most IRP listings, the order in which the labels usually appear, and the function done by the IRP at the specified label. An IRP listing is typically many pages long and contains more labels and IRP than indicated in Figure A-4 on page A-26. This figure is therefore intended to be used as a general directory into an IRP listing.

IRP Layout

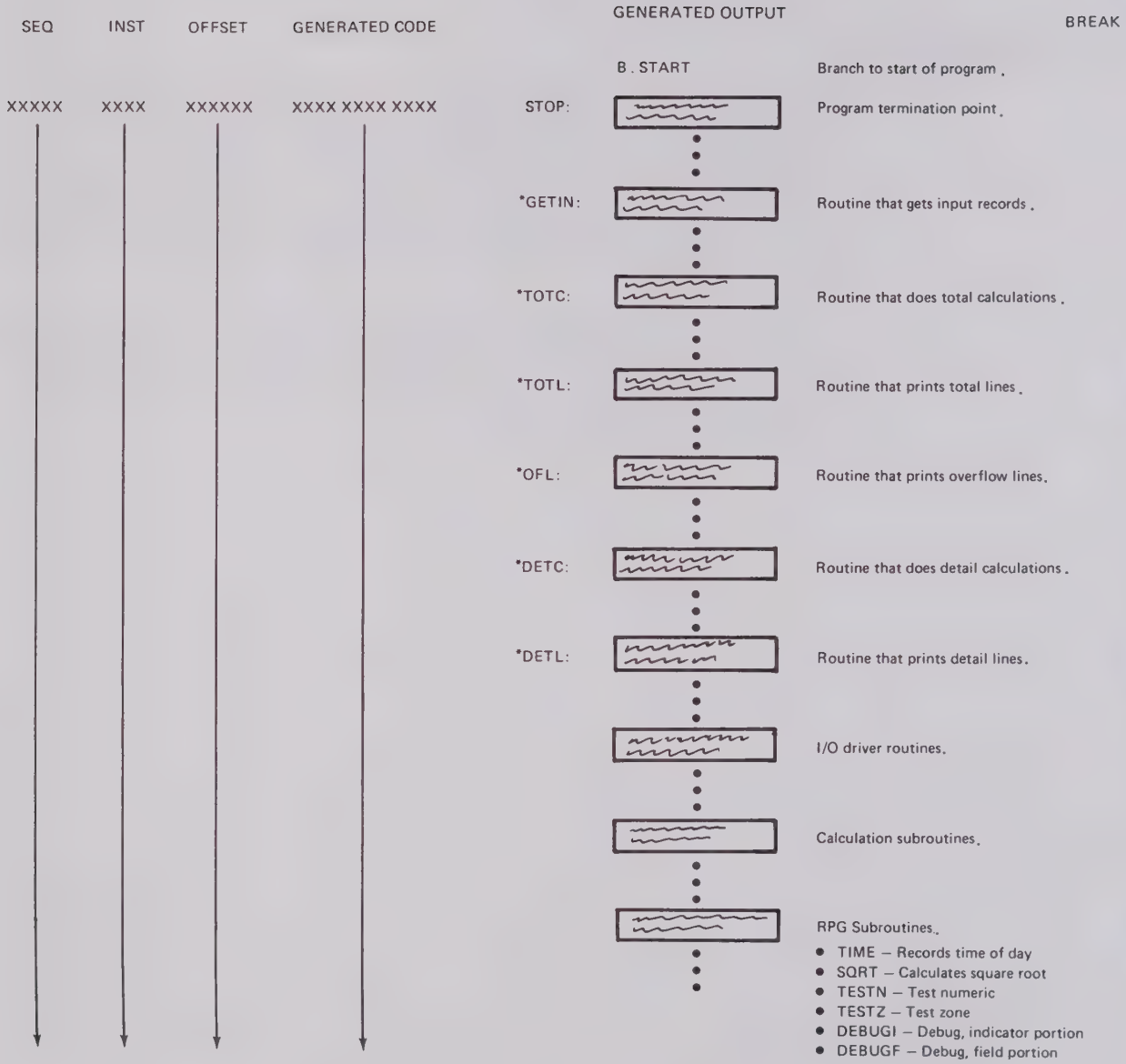


Figure A-4 (Part 1 of 2). IRP Layout

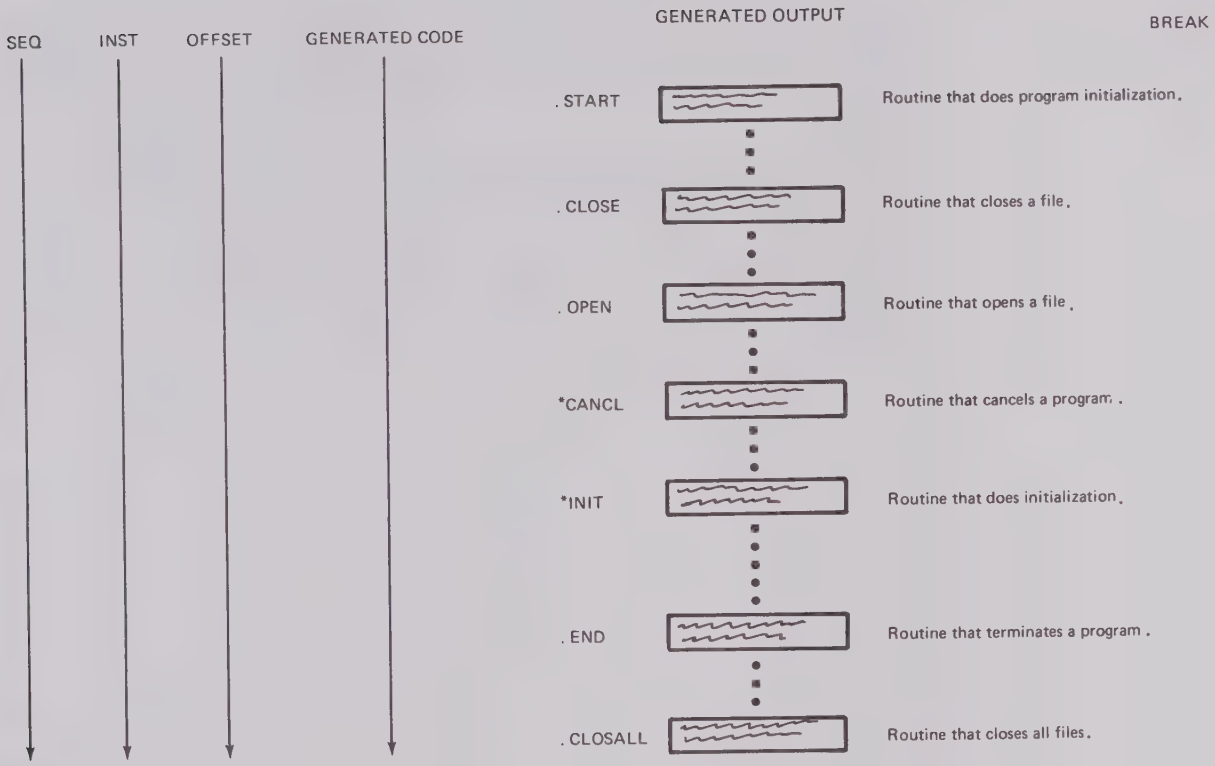


Figure A-4 (Part 2 of 2). IRP Layout

Auto Report Program

The auto report program consists of the phases listed in Table A-3. Those phases that have a U in the third column of Table A-3 process unconditionally, and those phases that have a C in the third column process only if they are required for the program being compiled. Figure A-5 on page A-29 shows the order in which the phases run.

The auto report program major data areas are a common area (XREGN), a field name table (FLDTBL), and two buffers (BUFFER1 and BUFFER2).

Table A-3 (Page 1 of 2). Auto Report Program Phases

Phase Name	Phase Description	Called: Unconditionally (U) Conditionally (C)
QRPT0000	Command interface phase	U
QRPT0001	Root controlling phase	U
QRPT0002	I/O control phase and /COPY function	U

Table A-3 (Page 2 of 2). Auto Report Program Phases

Phase Name	Phase Description	Called: Unconditionally (U) Conditionally (C)
QRPT0003	Diagnostic phase for *AUTO	C
QRPT0004	Diagnostic phase for *AUTO	C
QRPT0005	*AUTO generation phase	C
QRPT0008	Wrap-up phase	U

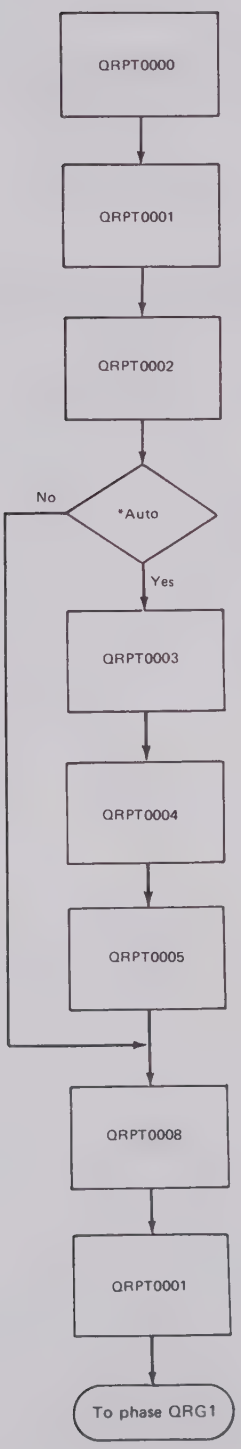


Figure A-5. Order of Processing of Auto Report Program Phases

Appendix B. Sample Program

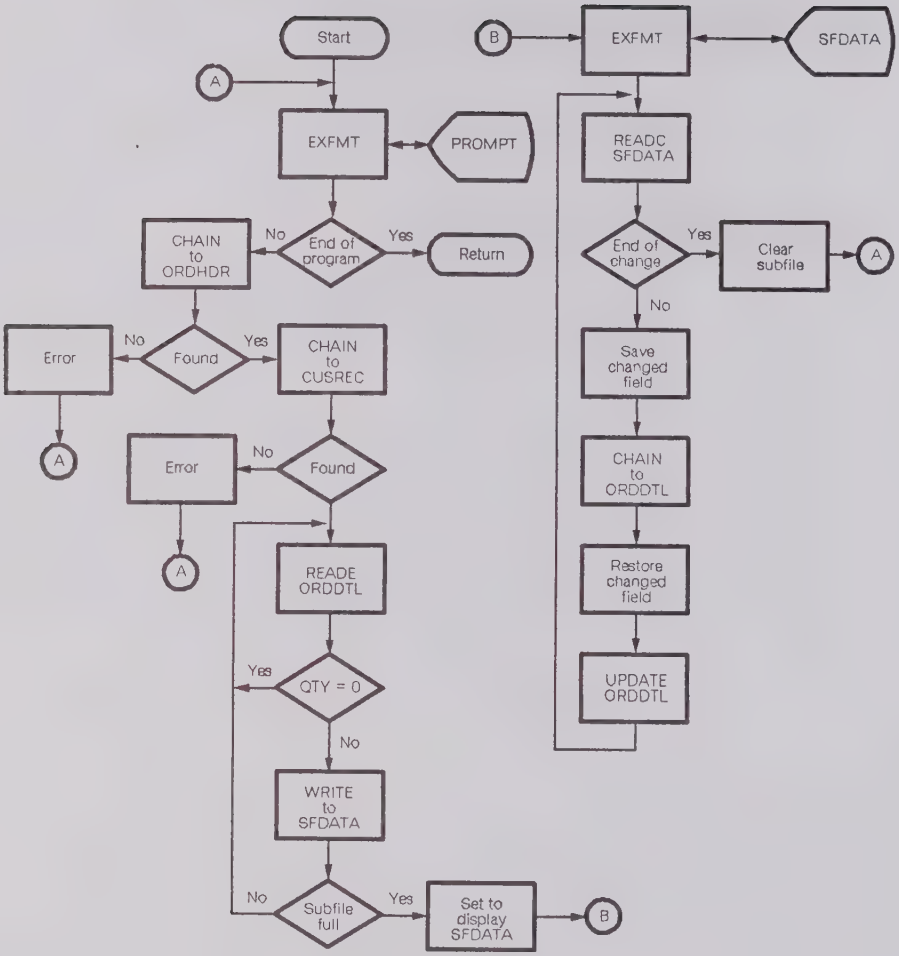
This appendix contains a sample RPG/400 program as it might be implemented on the AS/400 System.

Sample RPG Program (ORD400)

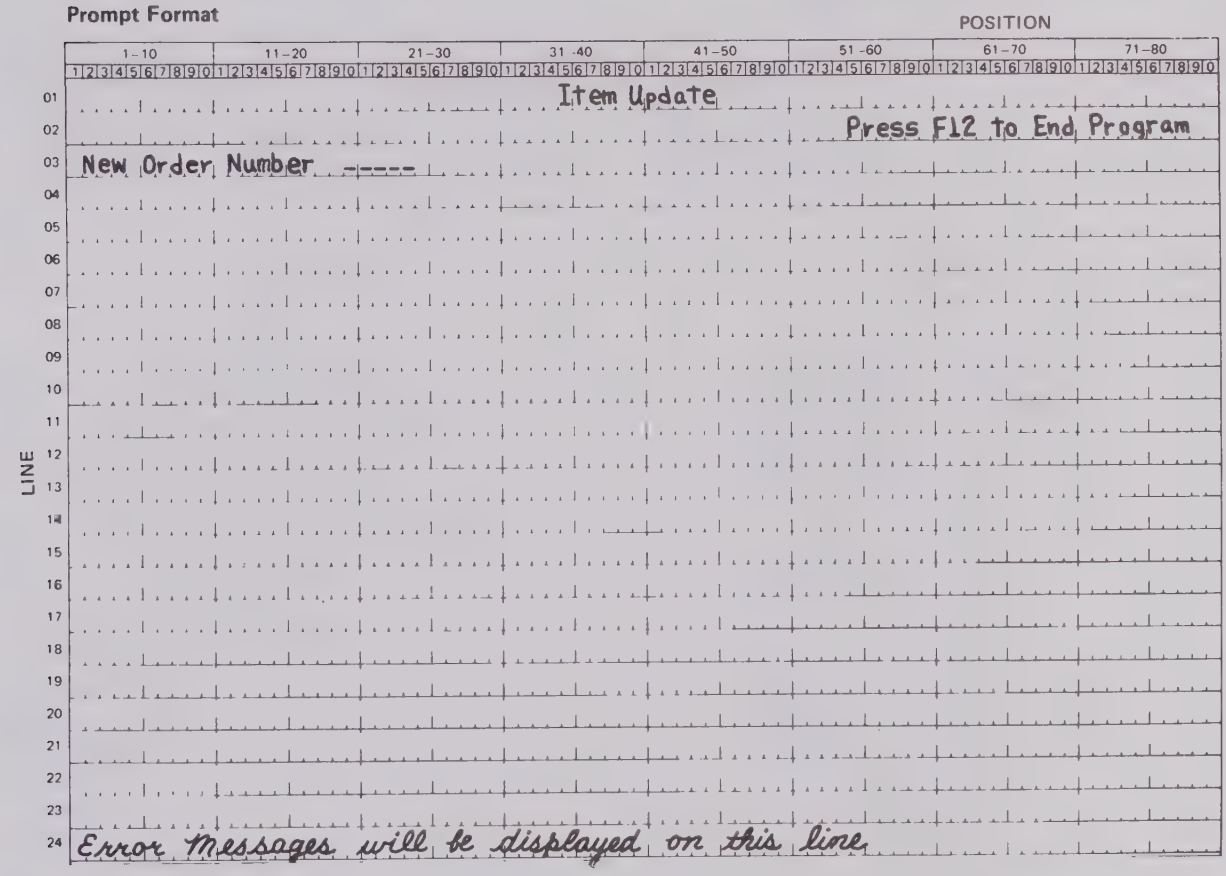
This sample is a hypothetical program only, designed for the sole purpose of illustrating RPG/400. The configuration of the the AS/400 System used in this example is assumed to include some number of IBM 5251 Display Stations (Model 11 or 12) as well as OS/400.

In this example, the shipping department needs to review the items on an order to post the changes to the quantity shipped if there is insufficient stock to fill the order. The clerk is first prompted to enter the order number to be reviewed. The program then displays the order, using data from the order detail record, as well as the customer number from the order header record and the customer name from the customer master record. After updating the appropriate items on the order, the clerk can either review additional orders or end the program.

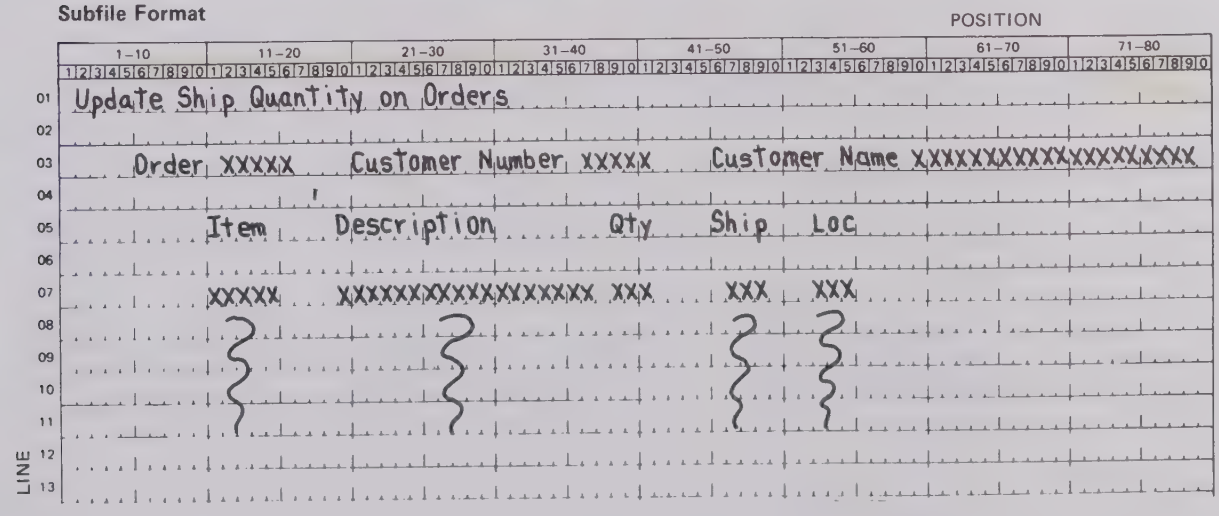
The program flow is shown on the following pages.


Start
A
EXFMT
PROMPT
CHAIN to ORDHDR
No
End of program
Yes
Return
Error
No
Found
Yes
CHAIN to CUSREC
A
Error
No
Found
A
READE ORDDTL
Yes
QTY = 0
No
WRITE to SFDATA
No
Subfile full
Yes
Set to display SFDATA
B
B
EXFMT
SFDATA
READC SFDATA
End of change
Yes
Clear subfile
A
No
Save changed field
CHAIN to ORDDTL
Restore changed field
UPDATE ORDDTL

Prompt Format



Subfile Format



This layout shows the displays that the clerk views. The clerk is prompted for the order number, and the program displays the order, using data from the order detail record, as well as the customer number from the order header record and the customer name from the customer master record.

Sample RPG Program (ORD400)

IBM International Business Machines Corporation — DATA DESCRIPTION SPECIFICATIONS — GX21-7754 UM/050* Printed in U.S.A.

File | Programmer | Date | Keying Instruction | Graphic Key | Description ORD400D | Page 1 of 2

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (B A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (R/O/I/B/H/M)	Location: Line	Pos	Functions
	A		*ORD400D -- ITEM UPDATE																
	A																		DSPSIZ(*DS3)
	A								R		PROMPT								
	A																		CA12(98 'End of Program')
	A																1	35	'Item Update'
	A																		DSPATR(HI)
	A																2	55	'Press F12 to End Program'
	A																3	2	'New Order Number'
	A										ORDER		5		ØI		3	20	
	A		91																ERRMSG('Order Number not Found' 91)
	A		92																ERRMSG('No Customer Record Found +
	A																		for this Order' 92)
	A		93																ERRMSG('# Lines in Order Exceeds +
	A																		60' 93)
	A		94																ERRMSG('No Lines for this Order' 94)
	A		95																ERRMSG('Update Failed -- Reenter +
	A																		Order Number' 95)
	A																		
	A																		
	A																		

*Number of sheets per pad may vary slightly

Subfiles are used in this program to illustrate how to handle multiple records of the same type on the display. The data description specifications for the ORD400D display file are shown here. Three display formats are used: PROMPT, SFDATA (the subfile record format), and SFCONTRL (the subfile control record format).

The PROMPT format requests the user to enter an order number. If there are no more orders to be entered, the user presses F12 to end the program. Error messages conditioned by indicators 91 through 95 are also displayed on this format.

The subfile record format SFDATA (shown on the following page) describes the fields used by the subfile. It is necessary only to describe the location of the first record in the display format. The subfile control record format specifies how many records are to be written to the display. The subfile record format description (SFDATA) must immediately precede the subfile control record format (SFCONTRL).

The SFCONTRL is used as:

- A subfile control record format
- A normal format to display customer name and number and redisplay the order number.

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754 UM/050* Printed in U.S.A.

File | Keying Instruction: Graphic | Key | Description: ORD400D | Page 2 of 2

Programmer | Date

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (B/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (B/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (B/O/I/B/H/M)	Location: Line	Pos	Functions
1 2 3 4 5	6	7	8	9 10	11 12	13 14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80	
	A							R		SFDATA									SFL
	A									ITEM		5		ØB		7	11		
	A																		DSPATR(PR)
	A																		CHGINPDFT
	A									DESCRP		18				7	2Ø		
	A									QTYORD		3		Ø		7	39		
	A									QTYSHP		3		ØB		7	47		
	A									WHSLOC		3				7	53		
	A							R		SFCONTRL									SFLCTL(SFDATA)
	A		65																SFLCLR
	A		N65																SFLDSP
	A		N65																SFLDSPCTL
	A																		SFLSIZ(6Ø)
	A																		SFLPAG(5)
	A															1	2	'Update Ship Quantity on Orders'	
	A																		DSPATR(HI)
	A															3	6	'Order'	
	A									CURORD		5		Ø		3	12		
	A															3	21	'Customer Number'	
	A									CUST		5				3	37		
	A															3	46	'Customer Name'	
	A									NAME		2Ø				3	6Ø		
	A															5	11	'Item'	
	A															5	2Ø	'Description'	
	A															5	39	'Qty'	
	A															5	46	'Ship'	
	A															5	53	'Loc'	
	A																		

The subfile data is displayed (SFLDSP keyword) if indicator 65 is off. If indicator 65 is on, the subfile is cleared (SFLCLR keyword). If indicator 65 is not on, the data in the SFCONTRL format is displayed (SFLDSPCTL keyword).

A limit of 60 line items for a particular order is specified by the SFLSIZ keyword, and a page size of 5 line items is specified by the SFLPAG keyword. Subfile support has paging capability that allows the program to first fill the subfile and then allows the display station user to roll pages up and down in the subfile without any program control.

Sample RPG Program (ORD400)

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754 UM/050* Printed in U.S.A.

File		Keying Instruction	Graphic								
Programmer	Date		Key								

Description	Page	of
CUSMSTL	1	1

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Name Type (blank/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (blank/A/P/S/B/A/S/X/Y/N/I/M/T)	Decimal Positions	Usage (blank/O/I/B/H/M)	Location: Line	Location: Pos	Functions
	A	**		LOGICAL							CUSMSTL	--	CUSTOMER		MASTER				FILE
	A																		UNIQUE
	A								R		CUSREC								PFILE(CUSMSTP)
	A																		TEXT('Customer Master Record')
	A										CUST								
	A										NAME								
	A										ADDR								
	A										CITY								
	A										STATE								
	A										ZIP								
	A										SRHCOD								
	A										CUSTYP								
	A										ARBAL								
	A										ORDBAL								
	A										LSTAMT								
	A										LSTDAT								
	A										CRDLMT								
	A										SLSYR								
	A										SLSLYR								
	A								K		CUST								

The data description specifications for the data base files used by this program are shown on the following pages. Included here are the data description specifications for the logical CUSMSTL (customer master) file, for the physical CUSMSTP (customer master) file, for the logical ORDFILL file (order file that contains the order header and order detail records), for the physical ORDHDRP (order header) file, and for the physical ORDDTLP (order detail) file. The physical file CUSMSTP is referenced by the logical file CUSMSTL. The physical files ORDHDRP and ORDDTLP are referenced by the logical file ORDFILL.

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754 UM 050*
Printed in U.S.A.

File		Keying Instruction	Graphic								
Programmer	Date		Key								

Description	Page	of

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name Not (N)	Condition Name Indicator	Condition Name Not (N)	Condition Name Indicator	Condition Name Not (N)	Condition Name Indicator	Name Type (R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (B A/P/S/B A/S/X-Y/N/I/N)	Decimal Positions	Usage (R/O/I/B/H/M)	Location: Line	Location: Pos	Functions
1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A	*	PHYSICAL						F		ILE CUSMSTP		-- CUSTOMER						MASTER FILE
	A								R		CUSMST								TEXT('Customer Master Record')
	A										BASDAT		6		Ø				EDTCDE(Y)
	A																		TEXT('Base Date Field')
	A										CUST		5						CHECK(ME)
	A																		COLHDG('Customer' 'Number')
	A										NAME		2Ø						COLHDG('Customer Name')
	A										ADDR	R							REFFLD(NAME)
	A																		COLHDG('Customer Address')
	A										CITY	R							REFFLD(NAME)
	A																		COLHDG('Customer City')
	A										STATE		2						CHECK(ME)
	A																		COLHDG('State')
	A										SRHCOD		6						CHECK(ME)
	A																		COLHDG('Search' 'Code')
	A																		TEXT('Customer Number Search Code')
	A										ZIP		5		Ø				CHECK(ME)
	A																		COLHDG('Zip' 'Code')
	A										CUSTYP		1		Ø				RANGE(1 5)
	A																		COLHDG('Cust' 'Type')
	A																		TEXT('Customer Type 1=Gov 2=Sch 3=Bus 4=Pvt 5=Oth')
	A										ARBAL		8		2				COLHDG('Accts Rec' 'Balance')
	A																		EDTCDE(J)
	A										ORDBAL	R							REFFLD(ARBAL)
	A																		COLHDG('A/R Amt In' 'Order File')
	A										LSTAMT	R							REFFLD(ARBAL)
	A																		COLHDG('Last' 'Amount' 'Paid')
	A																		TEXT('Last Amount Paid In A/R')
	A										LSTDAT	R							REFFLD(BASDAT)
	A																		COLHDG('Last' 'Date' 'Paid')
	A																		TEXT('Last Date Paid In A/R')
	A										CRDLMT	R							REFFLD(ARBAL)
	A																		COLHDG('Credit' 'Limit')
	A																		TEXT('Customer Credit Limit')
	A										SLSYR	R+	2						REFFLD(ARBAL)
	A																		COLHDG('Sales' 'This' 'Year')
	A																		TEXT('Customer Sales This Year')
	A										SLSLYR	R+	2						REFFLD(ARBAL)
	A																		COLHDG('Sales' 'Last' 'Year')
	A																		TEXT('Customer Sales Last Year')

IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754 UM/050* Printed in U.S.A.

File		Keying Instruction	Graphic							Description	Page	of
Programmer	Date		Key							ORDFILL	1	1

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not	Conditioning: Condition Name: Indicator	Name Type (blank/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (blank/A/P/S/B/A/S/X/Y/N/I/W)	Decimal Positions	Usage (blank/O/I/B/H/M)	Location: Line	Location: Pos	Functions
1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A	*	LOGICAL								ORDFILL	--	ORDER	F		I			LE OF ORDHDR AND ORDDTL
	A								R		ORDHDR								PFILE(ORDHDRP)
	A								K		ORDER								
	A								R		ORDDTL								PFILE(ORDDTLP)
	A								K		ORDER								
	A								K		ITEM								
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

IBM International Business Machines Corporation

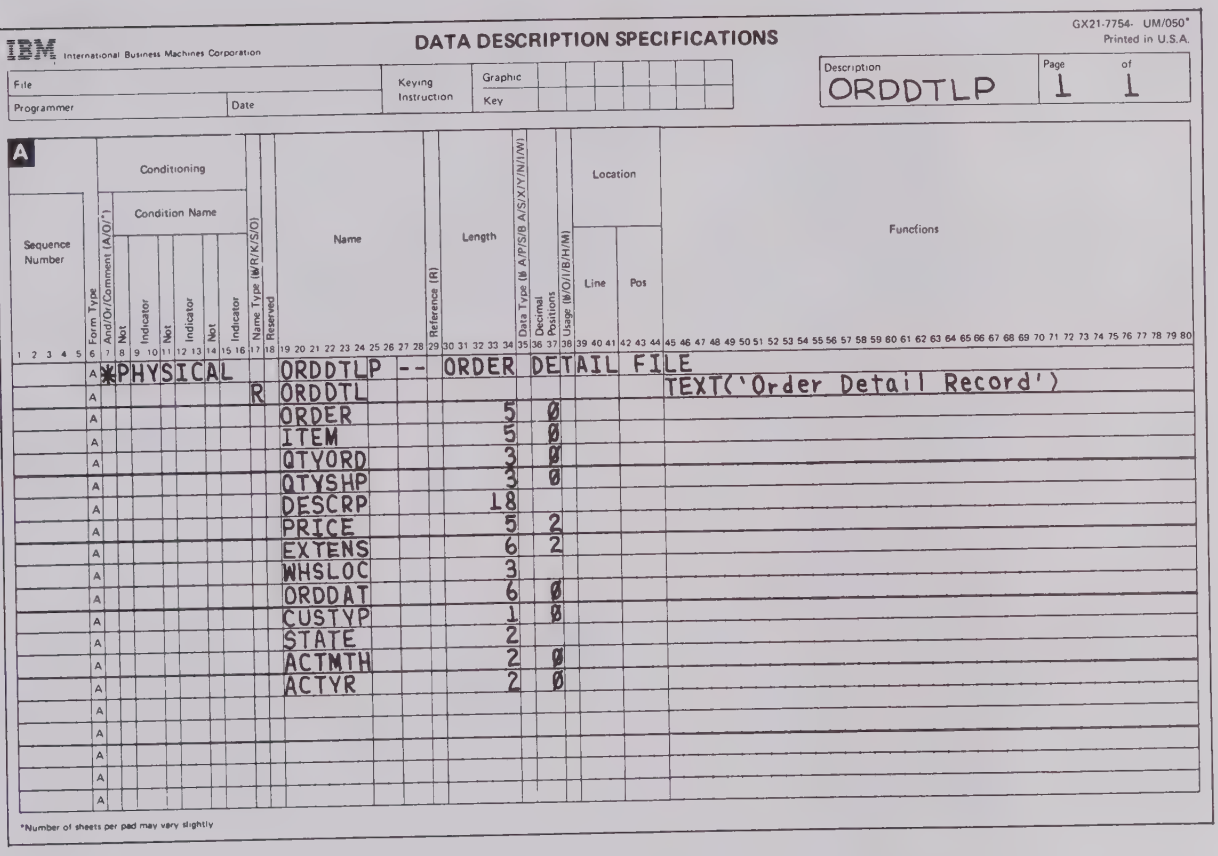
DATA DESCRIPTION SPECIFICATIONS

GX21-7754 UM/050* Printed in U.S.A.

File		Keying Instruction	Graphic							Description	Page	of
Programmer	Date		Key							ORDHDRP	1	1

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not	Conditioning: Condition Name: Indicator	Name Type (blank/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (blank/A/P/S/B/A/S/X/Y/N/I/W)	Decimal Positions	Usage (blank/O/I/B/H/M)	Location: Line	Location: Pos	Functions
1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A	*	PHYSICAL								ORDHDRP	--	ORDER	H		E			ADER FILE
	A								R		ORDHDR								TEXT('Order Header Record')
	A										CUST		5						
	A										ORDER		5		0				
	A										ORDDAT		6		0				
	A										CUSORD		15						
	A										SHPVIA		15						
	A										ORDSTS		1		0				
	A										ORDAMT		8		2				
	A										CUSTYP		1		0				
	A										INVNUM		5		0				
	A										PRTDAT		6		0				
	A										OPNSTS		1		0				
	A										TOTLIN		3		0				
	A										ACTMTH		2		0				
	A										ACTYR		2		0				
	A										STATE		2						
	A																		
	A																		
	A																		

*Number of sheets per pad may vary slightly



IBM International Business Machines Corporation

DATA DESCRIPTION SPECIFICATIONS

GX21-7754- UM/050* Printed in U.S.A.

File | Programmer | Date | Keying Instruction | Graphic Key

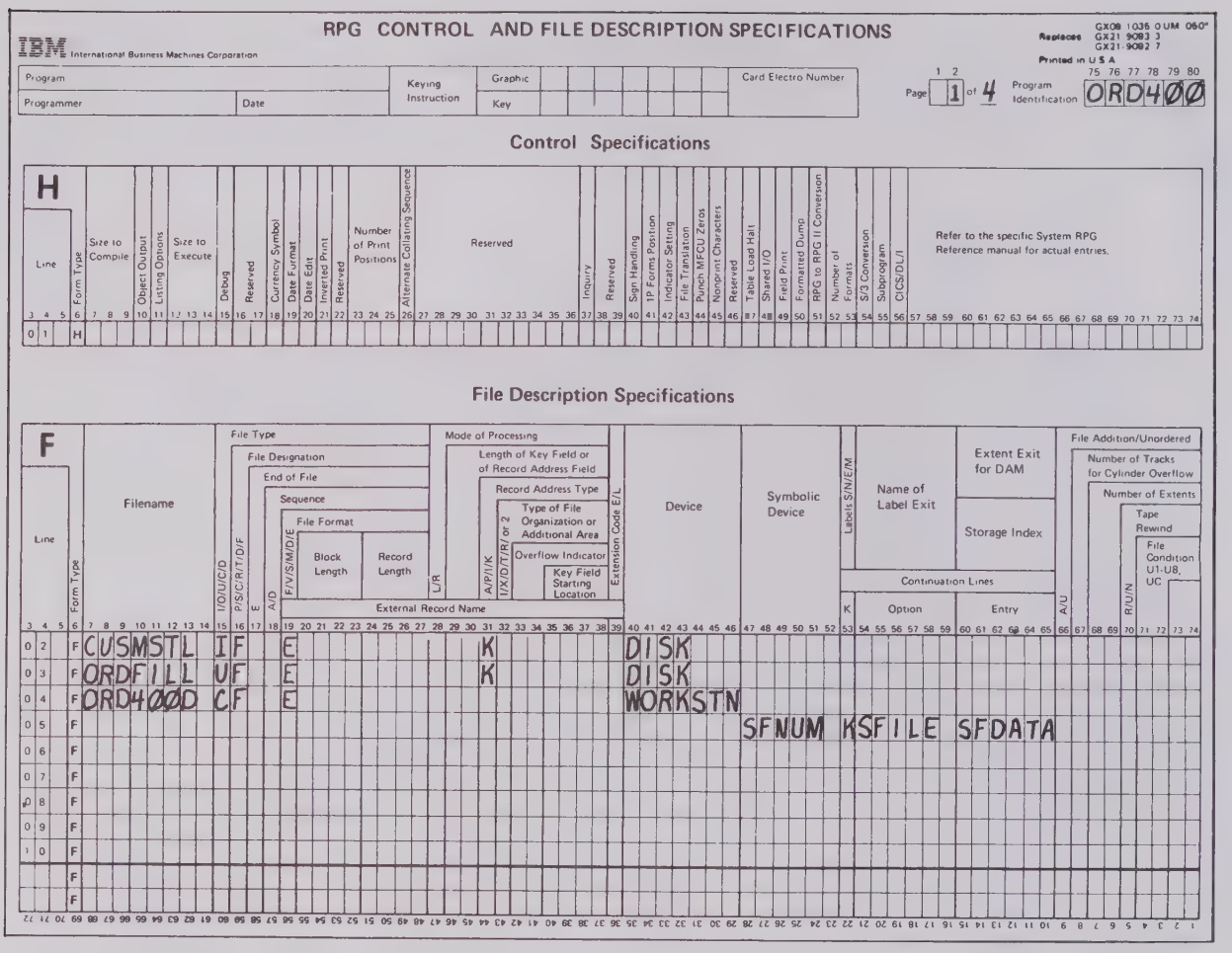
Description: ORDDTLP | Page 1 of 1

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not	Indicator	Not	Indicator	Not	Indicator	Name Type (R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (I/O/I/B/H/M)	Location: Line	Pos	Functions
1 2 3 4 5	6	7	8 9 10	11 12	13 14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80		
	A	*	PHYSICAL						ORDDTLP -- ORDER DETAIL FILE										
	A						R		ORDDTL										TEXT('Order Detail Record')
	A								ORDER		5		0						
	A								ITEM		5		0						
	A								QTYORD		3		0						
	A								QTYSHP		3		0						
	A								DESCRP		18								
	A								PRICE		5		2						
	A								EXTENS		6		2						
	A								WHSLOC		3								
	A								ORDDAT		6		0						
	A								CUSTYP		1		0						
	A								STATE		2								
	A								ACTMTH		2		0						
	A								ACTYR		2		0						
	A																		
	A																		
	A																		
	A																		
	A																		

*Number of sheets per pad may vary slightly

Sample RPG Program (ORD400)



RPG CONTROL AND FILE DESCRIPTION SPECIFICATIONS

IBM International Business Machines Corporation

GX08 1036 0 UM 060"
Replaces GX21 9083 3
GX21 9082 7
Printed in U.S.A.

Program | Keying Instruction | Graphic / Key | Card Electro Number
Programmer | Date

Page 1 of 4 | Program Identification ORD400

Control Specifications

H

Line | Form Type | Size to Compile | Object Output | Listing Options | Size to Execute | Debug | Reserved | Currency Symbol | Date Format | Date Edit | Inverted Print | Reserved | Number of Print Positions | Alternate Collating Sequence | Reserved | Inquiry | Reserved | Sign Handling | 1P Forms Position | Indicator Setting | File Translation | Punch MFCU Zeros | Nonprint Characters | Reserved | Table Load Halt | Shared I/O | Field Print | Formatted Dump | RPG to RPG II Conversion | Number of Formats | S/3 Conversion | Subprogram | CICS/DL/I | Refer to the specific System RPG Reference manual for actual entries.

File Description Specifications

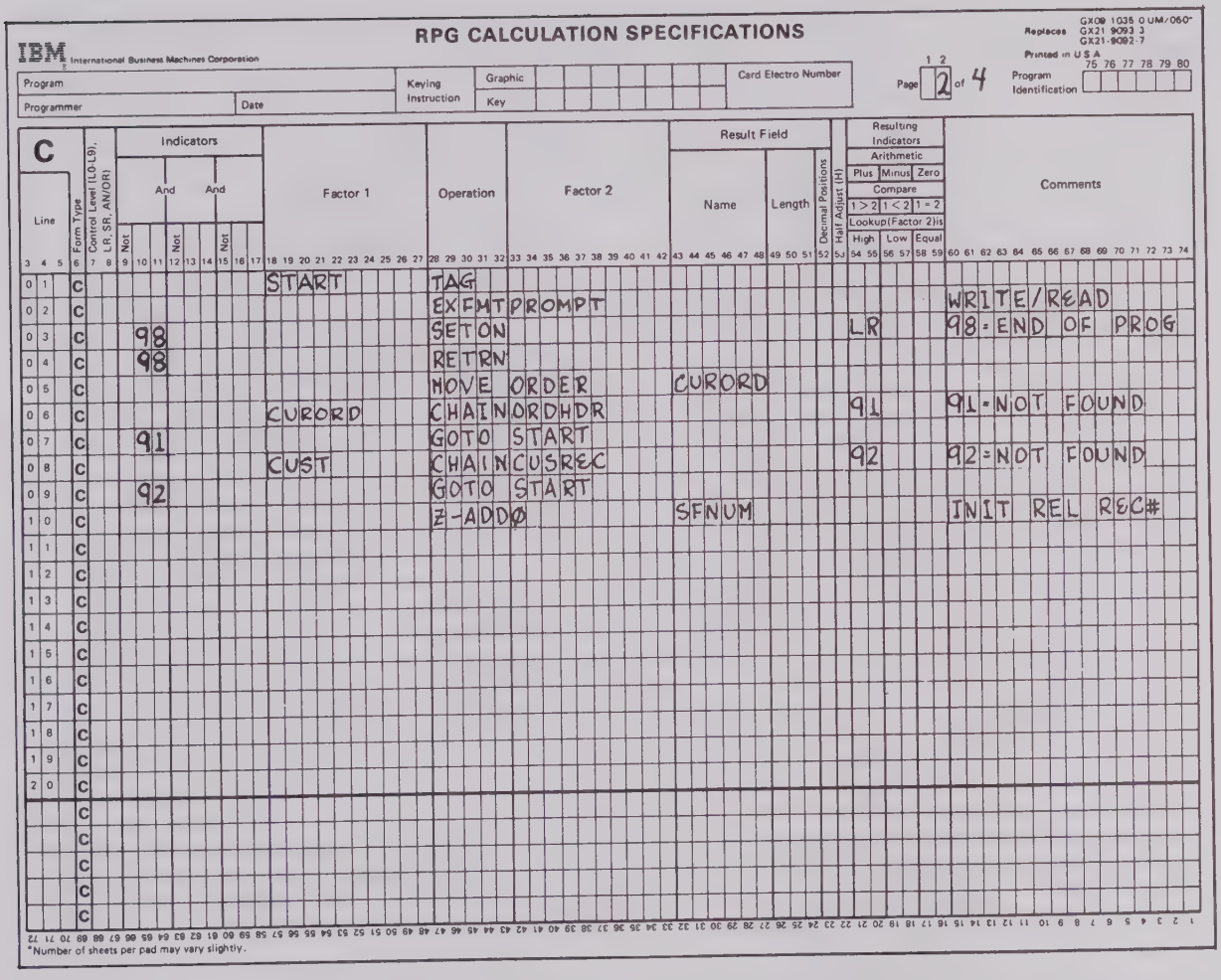
F

Line | Form Type | Filename | File Type | File Designation | End of File | Sequence | File Format | Block Length | Record Length | L/R | Mode of Processing | Length of Key Field or of Record Address Field | Record Address Type | Type of File Organization or Additional Area | Overflow Indicator | Key Field Starting Location | Extension Code | Device | Symbolic Device | Labels | Name of Label Exit | Extent Exit for DAM | Storage Index | File Addition/Unordered | Number of Tracks for Cylinder Overflow | Number of Extents | Tape Rewind | File Condition | Continuation Lines | K | Option | Entry | A/U | R/U/N | External Record Name

02	F	CUSMSTL	I	F		E					K							DISK																			
03	F	ORDFILL	U	F		E					K							DISK																			
04	F	ORD400D	C	F		E												WORKSTN																			
05	F																		SFNUM	KSFILE	SFDATA															
06	F																																			
07	F																																			
08	F																																			
09	F																																			
10	F																																			
	F																																			
	F																																			

The RPG/400 program uses the following three files:

- The CUSMSTL (customer master logical) file. The CUSMSTL file is a data base file (DISK in positions 40 through 46) that is processed as a full procedural file (F in position 16) with input only (I in position 15). This file is described externally (E in position 19) and is accessed by keys.
- The ORDFILL (order logical) file. The ORDFILL file is also described as a full procedural, data base file with a keyed access. The file is externally described. The ORDFILL file is defined as an update file (U in position 15). This logical file uses data from two physical files, ORDHDRP (order header file) and ORDDTLP (order detail file).
- The ORD400D display file. The ORD400D file is a display file (WORKSTN in positions 40 through 46) that is processed as a full procedural file and is described as a combined file (C in position 15). This file is externally described and contains the subfile SFDATA. A relative record number field (SFNUM) is defined for the SFDATA subfile.



RPG CALCULATION SPECIFICATIONS

IBM International Business Machines Corporation

Program | Programmer | Date | Keying Instruction | Graphic | Key | Card Electro Number

Page 2 of 4 | Program Identification

C

Line	Form Type	Control Level (L0-L9), LR, SR, AN/OR	Indicators: And / And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic: Plus / Minus / Zero; Compare: 1>2 / 1<2 / 1=2; Lookup(Factor 2)is: High / Low / Equal	Comments
01	C			START	TAG							
02	C				EXFMT PROMPT							WRITE/READ
03	C		98		SETON						LR	98=END OF PROG
04	C		98		RETRN							
05	C				MOVE ORDER		CURORD					
06	C			CURORD	CHAIN ORDHDR						91	91=NOT FOUND
07	C		91		GOTO START							
08	C			CUST	CHAIN CUSREC						92	92=NOT FOUND
09	C		92		GOTO START							
10	C				Z-ADD0		SFNUM					INIT REL REC#
11	C											
12	C											
13	C											
14	C											
15	C											
16	C											
17	C											
18	C											
19	C											
20	C											

*Number of sheets per pad may vary slightly.

The program sends the PROMPT format to the display station in the shipping department. The user responds by entering the order number to be reviewed and pressing the Enter key. If the user presses function key 12, indicator 98 is set on through the data description specifications for the PROMPT format, and the program ends.

If the user presses the Enter key, the program processes the CHAIN operation to the order header record to retrieve the customer number. If there is no record, an error occurs (indicator 91, which is specified in positions 54 and 55, is set on). The CHAIN operation on line 6 uses the order number (CURORD) as the search argument.

The program processes another CHAIN operation to the customer record (CUSREC) to retrieve the customer name.

The Z-ADD operation on line 10 initializes the SFNUM field, which is used as the relative record number (RECNO) field for the subfile.

RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Program | Punching Instruction | Graphic / Punch | Card Electro Number

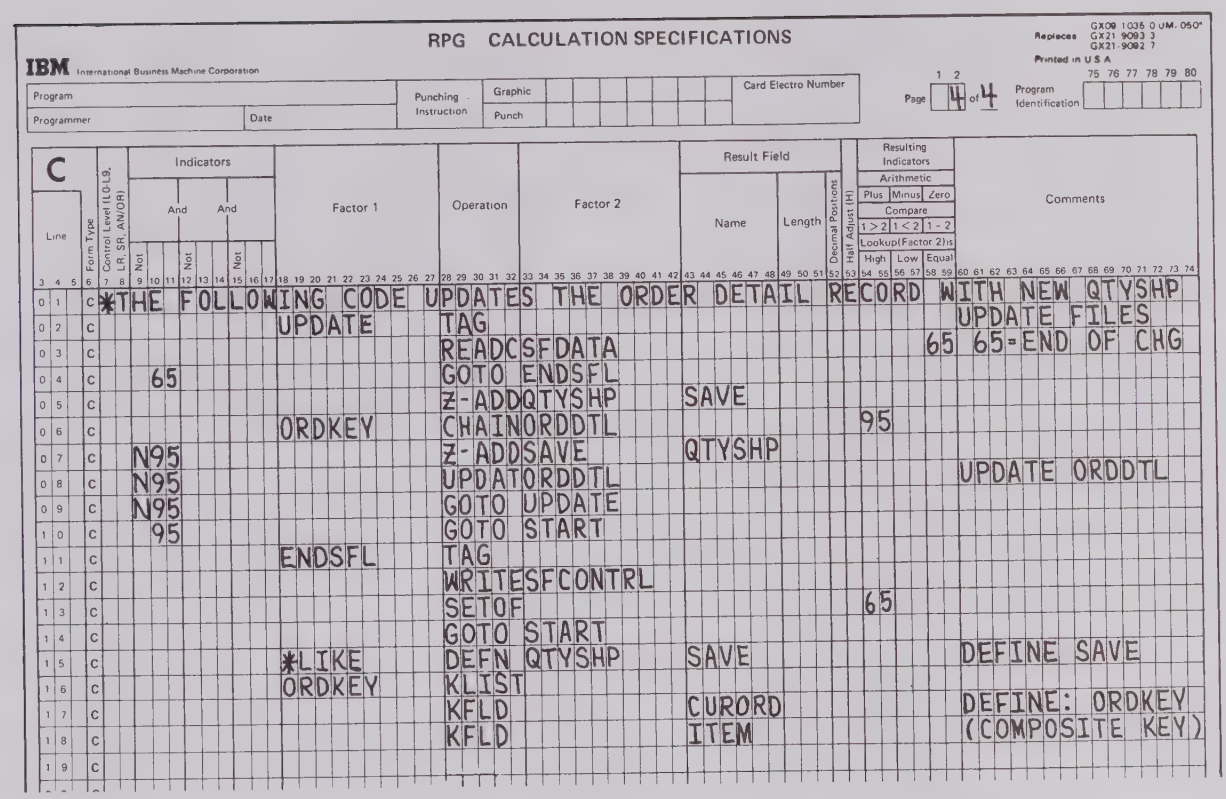
Programmer | Date

Page 3 of 4 | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus / Minus / Zero); Compare (1>2 / 1<2 / 1=2); Lookup(Factor 2)is (High / Low / Equal)	Comments
01	C		*THE FOLLOWING CODE		BUILDS THE SUBFILE							
02	C			BLDSF	TAG							BUILD SUBFILE
03	C			CURORD	READE ORDDTL							55 55=END OF ORDER
04	C		55		GOTO DSPSF							
05	C			QTYORD	IFNE Ø							IF ORD NOT CANC
06	C				ADD 1		SFNUM					NEXT REL REC
07	C				WRITE SFDATA							93 93=FULL SUBFIL
08	C				END							
09	C		N93		GOTO BLDSF							
10	C		*THE FOLLOWING CODE		DISPLAYS THE SUBFILE							
11	C			DSPSF	TAG							
12	C			SFNUM	COMP Ø							94 94=NO ITEMS
13	C		94		GOTO START							
14	C				Z-ADD1		SFNUM		30			
15	C				EXFMT SFCONTRL							
16	C											
17	C											
18	C											
19	C											
20	C											

The BLDSF routine reads the order detail records sequentially and fills the subfile with successive READE and WRITE operations. (Remember, in the logical file ORDFILL, records are accessed in order sequence so header records are logically intermixed with detail records and can be accessed sequentially by key—even though they are in different physical files.) If the quantity on order is zero, the record is bypassed.

If there are no more line items for this order (indicator 55 is on) or if the subfile is now full (indicator 93 is now on), the BLDSF routine ends.



RPG CALCULATION SPECIFICATIONS

IBM International Business Machine Corporation

Program | Programmer | Date | Punching Instruction | Graphic Punch | Card Electro Number | Page 4 of 4 | Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OA)	Indicators: And / And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus / Minus / Zero); Compare (1>2 / 1<2 / 1=2); Lookup(Factor 2)is (High / Low / Equal)	Comments
01	C		*THE FOLLOWING CODE UPDATES THE ORDER DETAIL RECORD WITH NEW QTYSHP									
02	C			UPDATE	TAG							UPDATE FILES
03	C				READCSFDATA						65	65=END OF CHG
04	C		65		GOTO ENDSFL							
05	C				Z-ADDQTYSHP		SAVE					
06	C			ORDKEY	CHAINORDDTL						95	
07	C		N95		Z-ADDSAVE		QTYSHP					
08	C		N95		UPDATORDDTL							UPDATE ORDDTL
09	C		N95		GOTO UPDATE							
10	C		95		GOTO START							
11	C			ENDSFL	TAG							
12	C				WRITESFCONTRL							
13	C				SETOF						65	
14	C				GOTO START							
15	C			*LIKE	DEFN QTYSHP		SAVE					DEFINE SAVE
16	C			ORDKEY	KLIST							
17	C				KFLD		CURORD					DEFINE: ORDKEY
18	C				KFLD		ITEM					(COMPOSITE KEY)
19	C											

The READC operation in the UPDATE routine returns to the program only those records in the subfile that were changed by the user. The changed field is temporarily saved while the old ORDDTL record is read into the program by the CHAIN operation. The ORDDTL record is then updated with the changed field.

If there are no more changed items for this order (indicator 65 is on), the program branches back to ENDSFL. The ENDSFL routine clears the subfile, and the program branches back to START.

If no record is found on the CHAIN (indicator 95 is on), the program branches back to START. An error message is then displayed, indicating that the update failed and requesting the user to reenter the ORDER key.

No input or output specifications are required in this program.

The data description listings and the RPG/400 compiler listing for sample program ORD400 are shown below and on the following pages:

```
5728RG1 R01M00  880715          IBM AS/400 RPG/400                 QGPL/ORD400       02/17/88  14:08:33      Page      1
Compiler . . . . . :   IBM AS/400 RPG/400
Command Options:
  Program  . . . . . :   QGPL/ORD400
  Source file  . . . . . :   *LIBL/ORD400
  Source member  . . . . . :   *PGM
  Source listing options . . . . . :   *SOURCE     *XREF       *GEN        *NODUMP     *NOSECLVL
  Generation options . . . . . :   *NOLIST     *NOXREF     *NOATR      *NODUMP     *NOOPTIMIZE
  Generation severity level  . . . :   0
  Print file . . . . . :   *LIBL/QSYSPRT
  Replace program  . . . . . :   *YES
  User profile . . . . . :   *USER
  Authority  . . . . . :   *CHANGE
  Text . . . . . :   *SRCMBRTXT
  Phase trace  . . . . . :   *NO
  Intermediate text dump . . . . . :   *NONE
  Snap dump  . . . . . :   *NONE
  Codelist . . . . . :   *NONE
  Ignore decimal data error  . . . :   *NO
Actual Program Source:
  File . . . . . :   ORD400
  Library  . . . . . :   *N
```

Figure B-1 (Part 1 of 6). Listing for ORD400 Program

```
5728RG1 R01M00  880715            IBM AS/400 RPG/400                 QGPL/ORD400       02/17/88  14:08:33      Page       2
SEQUENCE                                                                    IND   DO    LAST       PAGE   PROGRAM
NUMBER    *...1....+....2....+....3....+....4....+....5....+....6....+....7...*  USE   NUM   UPDATE     LINE   ID
                        S o u r c e   L i s t i n g                                                    *****
          H
   18200  FCUSMSTL IF  E           K        DISK                                                    02/15/88
           RECORD FORMAT(S):  LIBRARY QGPL FILE CUSMSTL.
                    EXTERNAL FORMAT CUSREC RPG NAME CUSREC
   18300  FORDFILL UF  E           K        DISK                                                    02/15/88
           RECORD FORMAT(S):  LIBRARY QGPL FILE ORDFILL.
                    EXTERNAL FORMAT ORDHDR RPG NAME ORDHDR
                    EXTERNAL FORMAT ORDDTL RPG NAME ORDDTL
   18400  FORD400D CF  E                    WORKSTN
   18500  F                                       SFNUM KSFILE SFDATA
           RECORD FORMAT(S):  LIBRARY QGPL FILE ORD400D.
                    EXTERNAL FORMAT PROMPT RPG NAME PROMPT
                    EXTERNAL FORMAT SFDATA RPG NAME SFDATA
                    EXTERNAL FORMAT SFCONTRL RPG NAME SFCONTRL
  A000000   INPUT  FIELDS FOR RECORD CUSREC FILE CUSMSTL FORMAT CUSREC.          CUSTOMER MASTER RECORD
  A000001                                          1   5 CUST                   CUSTOMER NUMBER
  A000002                                          6  25 NAME                   CUSTOMER NAME
  A000003                                         26  45 ADDR                   CUSTOMER ADDRESS
  A000004                                         46  65 CITY                   CUSTOMER CITY
  A000005                                         66  67 STATE                  STATE
  A000006                                       P 68  700ZIP                    ZIP CODE
  A000007                                         71  76 SRHCOD                 CUSTOMER NUMBER SEARCH CODE
  A000008                                       P 77  770CUSTYP                 CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OTH
  A000009                                       P 78  822ARBAL                  ACCTS REC BALANCE
  A000010                                       P 83  872ORDBAL                 A/R AMT IN ORDER FILE
  A000011                                       P 88  922LSTAMT                 LAST AMOUNT PAID IN A/R
  A000012                                       P 93  960LSTDAT                 LAST DATE PAID IN A/R
  A000013                                       P 97 1012CRDLMT                 CUSTOMER CREDIT LIMIT
  A000014                                       P 102 1072SLSYR                 CUSTOMER SALES THIS YEAR
  A000015                                       P 108 1132SLSLYR                CUSTOMER SALES LAST YEAR
  B000000   INPUT  FIELDS FOR RECORD ORDHDR FILE ORDFILL FORMAT ORDHDR.          ORDER HEADER RECORD
  B000001                                          1   5 CUST
  B000002                                       P  6   800RDER
  B000003                                       P  9  1200RDDAT
  B000004                                         13  27 CUSORD
  B000005                                         28  42 SHPVIA
  B000006                                       P 43  4300RDSTS
  B000007                                       P 44  4820RDAMT
  B000008                                       P 49  490CUSTYP
  B000009                                       P 50  520INVNUM
  B000010                                       P 53  560PRTDAT
  B000011                                       P 57  5700PNSTS
  B000012                                       P 58  590TOTLIN
  B000013                                       P 60  610ACTMTH
  B000014                                       P 62  630ACTYR
  B000015                                         64  65 STATE
  C000000   INPUT  FIELDS FOR RECORD ORDDTL FILE ORDFILL FORMAT ORDDTL.         ORDER DETAIL RECORD
```

Figure B-1 (Part 2 of 6). Listing for ORD400 Program

```
5728RG1 R01M00  880715            IBM AS/400 RPG/400                  QGPL/ORD400       02/17/88  14:08:33      Page       3
SEQUENCE                                                                    IND   DO    LAST       PAGE   PROGRAM
NUMBER    *...1....+....2....+....3....+....4....+....5....+....6....+....7...*  USE   NUM   UPDATE     LINE   ID
 C000001                                  P   1   300RDER
 C000002                                  P   4   60ITEM
 C000003                                  P   7   80QTYORD
 C000004                                  P   9  100QTYSHP
 C000005                                      11  28 DESCRP
 C000006                                  P  29  312PRICE
 C000007                                  P  32  352EXTENS
 C000008                                      36  38 WHSLOC
 C000009                                  P  39  4200RDDAT
 C000010                                  P  43  430CUSTYP
 C000011                                      44  45 STATE
 C000012                                  P  46  470ACTMTH
 C000013                                  P  48  490ACTYR
 D000000   INPUT  FIELDS FOR RECORD PROMPT FILE ORD400D FORMAT PROMPT.
 D000001                                   2   2 *IN91               Order Number Not Found
 D000002                                   3   3 *IN92               No Customer Record Found For This Order
 D000003                                   4   4 *IN93               # Lines in Order Exceeds 60
 D000004                                   5   5 *IN94               No Lines for This Order
 D000005                                   6   6 *IN95               Update Failed -- Reenter Order Number
 D000006                                   1   1 *IN98               End of Program
 D000007                                   7  110ORDER
 E000000   INPUT  FIELDS FOR RECORD SFDATA FILE ORD400D FORMAT SFDATA.
 E000001                                   1   50ITEM
 E000002                                   6  23 DESCRP
 E000003                                  24  260QTYORD
 E000004                                  27  290QTYSHP
 E000005                                  30  32 WHSLOC
 F000000   INPUT  FIELDS FOR RECORD SFCONTRL FILE ORD400D FORMAT SFCONTRL.
   18600  C           START     TAG
   18700  C                     EXFMTPROMPT                    WRITE/READ
   18800  C   98                SETON                    LR    98=END OF PROG  1
   18900  C   98                RETRN
   19000  C                     MOVE ORDER     CURORD
   19100  C           CURORD    CHAINORDHDR              91    91=NOT FOUND    1
   19200  C   91                GOTO START
   19300  C           CUST      CHAINCUSREC              92    92=NOT FOUND    1
   19400  C   92                GOTO START
   19500  C                     Z-ADD0         SFNUM           INIT REL REC#
   19600  C*THE FOLLOWING CODE BUILDS THE SUBFILE
   19700  C           BLDSF     TAG                            BUILD SUBFILE
   19800  C           CURORD    READEORDDTL                  5555=END OF ORDER     3
   19900  C   55                GOTO DSPSF
   20000  C           QTYORD    IFNE 0                         IF ORD NOT CANC        B001
   20100  C                     ADD  1         SFNUM           NEXT REL REC            001
   20200  C                     WRITESFDATA                  93 93=FULL SUBFIL     3   001
   20300  C                     END                                                    E001
   20400  C  N93                GOTO BLDSF
   20500  C*THE FOLLOWING CODE DISPLAYS THE SUBFILE
   20600  C           DSPSF     TAG
   20700  C           SFNUM     COMP 0                       94 94=NO ITEMS        3
   20800  C   94                GOTO START
   20900  C                     Z-ADD1         SFNUM   30
   21000  C                     EXFMTSFCONTRL
   21100  C*THE FOLLOWING CODE UPDATES THE ORDER DETAIL RECORD WITH NEW QTYSHP
```

Figure B-1 (Part 3 of 6). Listing for ORD400 Program

```
5728RG1 R01M00  880715            IBM AS/400 RPG/400                    QGPL/ORD400       02/17/88  14:08:33      Page       4
SEQUENCE                                                                      IND    DO    LAST       PAGE   PROGRAM
NUMBER    *...1....+....2....+....3....+....4....+....5....+....6....+....7...* USE    NUM   UPDATE     LINE   ID
   21200  C           UPDATE    TAG                              UPDATE FILES
   21300  C                     READCSFDATA                    65 65=END OF CHG      3
   21400  C   65                GOTO ENDSFL
   21500  C                     Z-ADDQTYSHP    SAVE
   21600  C           ORDKEY    CHAINORDDTL              95                    1
   21700  C  N95                Z-ADDSAVE      QTYSHP
   21800  C  N95                UPDATORDDTL                     UPDATE ORDDTL
   21900  C  N95                GOTO UPDATE
   22000  C   95                GOTO START
   22100  C           ENDSFL    TAG
   22200  C                     WRITESFCONTRL
   22300  C                     SETOF                    65                    1
   22400  C                     GOTO START
   22500  C           *LIKE     DEFN QTYSHP    SAVE             DEFINE SAVE
   22600  C           ORDKEY    KLIST
   22700  C                     KFLD           CURORD           DEFINE: ORDKEY
   22800  C                     KFLD           ITEM             (COMPOSITE KEY)
 G000000   OUTPUT  FIELDS FOR RECORD ORDDTL FILE ORDFILL FORMAT ORDDTL.       ORDER DETAIL RECORD
 G000001                         ORDER      3P PACK  5,0
 G000002                         ITEM       6P PACK  5,0
 G000003                         QTYORD     8P PACK  3,0
 G000004                         QTYSHP    10P PACK  3,0
 G000005                         DESCRP    28  CHAR   18
 G000006                         PRICE     31P PACK  5,2
 G000007                         EXTENS    35P PACK  6,2
 G000008                         WHSLOC    38  CHAR    3
 G000009                         ORDDAT    42P PACK  6,0
 G000010                         CUSTYP    43P PACK  1,0
 G000011                         STATE     45  CHAR    2
 G000012                         ACTMTH    47P PACK  2,0
 G000013                         ACTYR     49P PACK  2,0
 H000000   OUTPUT  FIELDS FOR RECORD PROMPT FILE ORD400D FORMAT PROMPT.
 H000001                         *IN91      1  CHAR    1                  Order Number Not Found
 H000002                         *IN92      2  CHAR    1                  No Customer Record Found For This Order
 H000003                         *IN93      3  CHAR    1                  # Lines in Order Exceeds 60
 H000004                         *IN94      4  CHAR    1                  No Lines for This Order
 H000005                         *IN95      5  CHAR    1                  Update Failed -- Reenter Order Number
 I000000   OUTPUT  FIELDS FOR RECORD SFDATA FILE ORD400D FORMAT SFDATA.
 I000001                         ITEM       5  ZONE  5,0
 I000002                         DESCRP    23  CHAR   18
 I000003                         QTYORD    26  ZONE  3,0
 I000004                         QTYSHP    29  ZONE  3,0
 I000005                         WHSLOC    32  CHAR    3
 J000000   OUTPUT  FIELDS FOR RECORD SFCONTRL FILE ORD400D FORMAT SFCONTRL.
 J000001                         *IN65      1  CHAR    1
 J000002                         CURORD     6  ZONE  5,0
 J000003                         CUST      11  CHAR    5
 J000004                         NAME      31  CHAR   20
        *****  E N D   O F   S O U R C E  *****
       A d d i t i o n a l   D i a g n o s t i c   M e s s a g e s
5728RG1 R01M00  880715            IBM AS/400 RPG/400                    QGPL/ORD400       02/17/88  14:08:33      Page       5
            K e y   F i e l d   I n f o r m a t i o n
                PHYSICAL    LOGICAL
     FILE/RCD   FIELD       FIELD       ATTRIBUTES
  01 CUSMSTL
        CUSREC
                CUST                    CHAR    5
  02 ORDFILL
        COMMON KEY FIELD ATTRIBUTES
                                        PACK  5,0  SIGNED
        ORDHDR
                ORDER                   PACK  5,0  SIGNED
        ORDDTL
                ORDER                   PACK  5,0  SIGNED
                ITEM                    PACK  5,0  SIGNED
```

Figure B-1 (Part 4 of 6). Listing for ORD400 Program

```
5728RG1 R01M00  880715           IBM AS/400 RPG/400                 QGPL/ORD400       02/17/88  14:08:33      Page       5
                          C r o s s   R e f e r e n c e
File and Record References:
       FILE/RCD    DEV/RCD     REFERENCES (D=DEFINED)
   01  CUSMSTL     DISK          18200D
         CUSREC                  18200D A000000    19300
   02  ORDFILL     DISK          18300D
         ORDDTL                  18300D C000000    19800    21600    21800
                                G000000
         ORDHDR                  18300D B000000    19100
   03  ORD400D     WORKSTN       18400D
         PROMPT                  18400D D000000    18700  H000000
         SFCONTRL                18400D F000000    21000    22200  J000000
         SFDATA                  18400D E000000    20200    21300  I000000
Field References:
       FIELD       ATTR        REFERENCES (M=MODIFIED D=DEFINED)
       *IN65       A(1)    J000001D
       *IN91       A(1)    D000001  H000001D
       *IN92       A(1)    D000002  H000002D
       *IN93       A(1)    D000003  H000003D
       *IN94       A(1)    D000004  H000004D
       *IN95       A(1)    D000005  H000005D
* 7031  *IN98       A(1)    D000006
       *LIKE                 22500
       ACTMTH      P(2,0)  B000013D C000012D G000012D
       ACTYR       P(2,0)  B000014D C000013D G000013D
* 7031  ADDR        A(20)   A000003D
* 7031  ARBAL       P(8,2)  A000009D
       BLDSF       TAG       19700D   20400
* 7031  CITY        A(20)   A000004D
* 7031  CRDLMT      P(8,2)  A000013D
       CURORD      P(5,0)    19000M   19100    19800    22700  J000002D
* 7031  CUSORD      A(15)   B000004D
       CUST        A(5)    A000001D B000001D   19300  J000003D
       CUSTYP      P(1,0)  A000008D B000008D C000010D G000010D
       DESCRP      A(18)   C000005D E000002D G000005D I000002D
       DSPSF       TAG       19900    20600D
       ENDSFL      TAG       21400    22100D
       EXTENS      P(6,2)  C000007D G000007D
* 7031  INVNUM      P(5,0)  B000009D
       ITEM        P(5,0)  C000002D E000001D   22800  G000002D I000001D
* 7031  LSTAMT      P(8,2)  A000011D
* 7031  LSTDAT      P(6,0)  A000012D
       NAME        A(20)   A000002D J000004D
* 7031  OPNSTS      P(1,0)  B000011D
* 7031  ORDAMT      P(8,2)  B000007D
```

Figure B-1 (Part 5 of 6). Listing for ORD400 Program

```
5728RG1 R01M00  880715            IBM AS/400 RPG/400                  QGPL/ORD400       02/17/88  14:08:33      Page       7
* 7031  ORDBAL      P(8,2)  A000010D
        ORDDAT      P(6,0)  B000003D C000009D G000009D
        ORDER       P(5,0)  B000002D C000001D D000007D   19000  G000001D
        ORDKEY      KLIST     21600    22600D
* 7031  ORDSTS      P(1,0)  B000006D
        PRICE       P(5,2)  C000006D G000006D
* 7031  PRTDAT      P(6,0)  B000010D
        QTYORD      P(3,0)  C000003D E000003D   20000  G000003D I000003D
        QTYSHP      P(3,0)  C000004D E000004D   21500    21700M   22500
                            G000004D I000004D
        SAVE        P(3,0)    21500M   21700    22500D
        SFNUM       P(3,0)    18400    19500M   20100M   20700    20900D
* 7031  SHPVIA      A(15)   B000005D
* 7031  SLSLYR      P(10,2) A000015D
* 7031  SLSYR       P(10,2) A000014D
* 7031  SRHCOD      A(6)    A000007D
        START       TAG       18600D   19200    19400    20800    22000
                              22400
        STATE       A(2)    A000005D B000015D C000011D G000011D
* 7031  TOTLIN      P(3,0)  B000012D
        UPDATE      TAG       21200D   21900
        WHSLOC      A(3)    C000008D E000005D G000008D I000005D
* 7031  ZIP         P(5,0)  A000006D
        0           LITERAL   19500    20000    20700
        1           LITERAL   20100    20900
 Indicator References:
        INDICATOR  REFERENCES (M=MODIFIED D=DEFINED)
        *IN        D000001  D000002  D000003  D000004  D000005  D000006
                   H000001D H000002D H000003D H000004D H000005D J000001D
        LR           18800M
        55           19800M   19900
        65           21300M   21400    22300M J000001D
        91         D000001    19100M   19200  H000001D
        92         D000002    19300M   19400  H000002D
        93         D000003    20200M   20400  H000003D
        94         D000004    20700M   20800  H000004D
        95         D000005    21600M   21700    21800    21900    22000
                   H000005D
        98         D000006    18800    18900
     * * * * *  E N D  O F  C R O S S  R E F E R E N C E  * * * * *
5728RG1 R01M00  880715            IBM AS/400 RPG/400                  QGPL/ORD400       02/17/88  14:08:33      Page       8
                      M e s s a g e  S u m m a r y
* QRG7031 Severity:  00   Number:   20
         Message . . . . :   The Name or indicator is not referenced.
     * * * * *  E N D  O F  M E S S A G E  S U M M A R Y  * * * * *
5728RG1 R01M00  880715            IBM AS/400 RPG/400                  QGPL/ORD400       02/17/88  14:08:33      Page       9
                        F i n a l  S u m m a r y
 Message Count:  (by Severity Number)
           TOTAL    00     10     20     30     40     50
            20      20      0      0      0      0      0
 Program Source Totals:
    Records . . . . . . . . . . :  47
    Specifications  . . . . . . :  44
    Table Records . . . . . . . :  0
    Comments  . . . . . . . . . :  3
 PRM has been called.
 Program ORD400 is placed in library QGPL. 00 highest Error-Severity-Code.
         * * * * *  E N D  O F  C O M P I L A T I O N  * * * * *
```

Figure B-1 (Part 6 of 6). Listing for ORD400 Program

Appendix C. Differences Between AS/400 RPG/400 and AS/400 RPG II System/36-Compatible

This appendix contains brief information on the AS/400 System enhancements to RPG/400. The topics covered are:

- Language enhancements
- New compilation listing functions
- Enhanced debug facilities
- Other support.

Language Enhancements

The RPG/400 language enhancements to AS/400 RPG II System/36-Compatible:

- *Externally described data:* Fields of a file are described to the OS/400 through data description specifications (DDS). The advantage of externally described data is that the fields of the file need be described only once to the OS/400 and not be described for each program that uses the file; if the file description changes, the programmer can change its description in one place and then recompile the programs that use it.

  The following file processing operation codes are new to RPG/400:

  - DELET (Delete Record)
  - SETGT (Set Greater Than)
  - UPDAT (Modify Existing Record)
  - WRITE (Create New Records).

  The following control operation codes are new to RPG/400:

  - CLOSE (Close Files)
  - FEOD (Force End of Data)
  - OPEN (Open File for Processing).

- *Work station support:* Allows the specification of the RPG/400 device name, WORKSTN, which is used for input and output from the display work station. The operation codes that support direct control over specific work station formats include:

  - EXFMT (Execute Format)
  - WRITE (Create New Records).

  In addition, RPG/400 supports the subfile capability in WORKSTN support with the following operation codes:

  - CHAIN (Random Retrieval from a Subfile)
  - READC (Read Next Modified Record)
  - UPDAT (Modify Existing Record)
  - WRITE (Create New Records).

  See Chapter 9, Using WORKSTN files for further information on the WORKSTN device.

- *Data structures:* RPG/400 supports two additional data structures:
  - Program status data structure: Provides program exception/error information to the program
  - Externally described data structures.
- *Multiple occurrence data structures:* This capability allows a data structure to appear n times in a program. See "Multiple Occurrence Data Structure" on page 10-9 for further information on multiple occurrence data structures.
- *Data area operations:* The IN (Retrieve a Data Area) and OUT (Write a Data Area) operations allow you to access a data area and optionally allow you to lock or unlock a data area. The UNLCK operation (Unlock a Data Area) unlocks one or all locked data areas in a program.
- *CABxx (compare and branch) operation:* Allows the programmer to do a compare and branch in one operation and eliminates the need to set and test resulting indicators.
- *Indicators referred to as data (*IN, *INxx):* Allow the programmer an alternative method of referring to and manipulating indicators. The indicator array of indicators 01 through 99 (*IN) and the indicator field (*INxx) reduce coding and provide a simplified approach to many program processing requirements. See Chapter 12, "Using Indicators" for information on indicators referred to as data.
- *ANDxx/ORxx operations:* Allow the programmer to specify a more complex decision condition than a simple A to B comparison with the IFxx, DOUxx, and DOWxx operations.
- *Exception/error handling:* Allows the programmer to control the program logic, by using the program exception/error subroutine *PSSR if a program exception/errors occur while the program is running. See "Exception/Error Handling" on page 4-35 for detailed information on program exception/error handling.
- *TESTN (Test Numeric) operation:* Allows the programmer to validity check a character field to ensure that it contains zoned decimal digits and blanks.
- *Overflow indicators:* Indicators 01 through 99 can be used as overflow indicators on both program described and externally described PRINTER files (in addition to 0A through 0G and 0V for program described files). See Chapter 6, "General File Considerations" for further information on overflow indicators.
- *Resulting indicators with MOVE and MOVEL operations:* Resulting indicators can be specified on MOVE and MOVEL statements. This eliminates the need for additional operations to check for blank, zero, or plus/minus conditions.
- *Figurative constants:* Allow the use of 3 additional RPG/400 reserved words that can be specified without specifying length and decimal positions because the implied length and decimal positions of a figurative constant are the same as that of the associated field. The three are:
  - *HIVAL
  - *LOVAL
  - *ALL'X..'

- *Commitment control:* Allows you to group file operations by using the COMIT and ROLBK operation codes. See Chapter 7, "Commitment Control" for information.
- *KLIST (Define a Composite Key)/KFLD (Define Parts of a Key) operations:* Allow the programmer to indicate the name by which a composite key may be specified and the fields that comprise the composite key.
- *SPECIAL file with PLIST operation:* Allows the programmer to specify an input/output device that is not directly supported by RPG/400. The programmer can add additional parameters to the RPG-created parameter list with the use of the PLIST and PARM operation codes. See "Special File" on page 6-29 for information on the SPECIAL device.
- *User-defined edit codes (5 through 9):* Allow for unique customer or national oriented editing. The user-defined edit codes are defined to the AS/400 System See Chapter 14, "Editing Numeric Fields" for further information on the user-defined edit codes.
- *Printer control (PRTCTL) option:* The programmer can (1) dynamically specify space/skip operations instead of using values on the output specifications, and (2) access the current line value within the program. This option is allowed only with program described files. See Chapter 6, "General File Considerations" for further information on the PRTCTL options.
- *DSPLY (Display Function) operation:* Allows access to a display device without the use of a display device file and allows access to the message handler.
- *SEQ files:* Allow you to perform sequential input/output to any sequentially organized file, such as data base, diskette, tape, savefile, or printer file. The actual device used is specified by an AS/400 Control Program Facility override command.
- RPG/400 supports numeric variables up to and including 30 digits. The maximum number of decimal digits allowed remains 9.
- *The compiler /COPY function:* Allows the merging of members from more than one source file during a compile. No sorting or modification of records can be done.

Note: For more information on the above mentioned operations, see the *Languages: RPG/400 Reference.*

Appendix D. Data Communication

AS/400 RPG/400 allows data communication through the WORKSTN file using ICF. There are no RPG/400 operations or specifications unique to data communication. The kinds of data communication supported through the WORKSTN file using ICF include APPC, Asynchronous, BSCEL, and SNUF. The WORKSTN file used for data communication must be defined as a full procedural (F in position 16 of the file description specifications). Here is a list of some operation codes and corresponding data communication functions supported by the WORKSTN file:

Operation Codes	Data Communication Functions
OPEN (Open File for Processing)	Open (Input and Output), error recovery
CLOSE (Close Files)	Close (Permanent), error recovery
EXFMT (Execute Format)	Write/Read (Wait)
READ (Read a Record)	Read (Wait)
WRITE (Create New Records)	Write (Wait)
ACQ (Acquire)	Acquire a device, error recovery
REL (Release)	Release a device, error recovery

For more information on remote communication, see the *Communications Programmer's Guide*. For information on RPG/400 operations, see *Languages: RPG/400 Reference*.

Exception/Error Handling with ICF Files

When a program has a run-time error, you may elect to cancel the program (see "Exception/Error Handling" on page 4-35). All of the program's files are closed abnormally. For communications files, the other end of the communications line is notified that there has been a failure and the communication has ended abnormally. For a shared communications file, the notification is sent when the last program closes the file.

Instead of cancelling a program, you may elect to continue processing (for example, in an error handling subroutine). It is your responsibility either to recover from the error, or else to notify the other party in the communication of the error by writing a record format with the FAIL keyword enabled (see the *Communications Programmer's Guide* for information on the FAIL keyword). The use of the FAIL keyword also ensures that when the file is closed, the time allowed for a response from the other end is short.

Communications Error Recovery

You may be able to recover from a device communications error, when using a multiple device file, by reissuing the ACQ (Acquire) operation for the device in error.

You may be able to recover from a file communications error by processing a CLOSE operation followed by an OPEN operation for the file in error. With shared files, the program must be closed for all the programs sharing the file, and then opened again.

For further information, see the *Communications: Distributed Data Management User's Guide.*

Appendix E. RPG Restrictions

Function	Restriction
AN/OR lines (positions 7 and 8 of calculation specifications)	Maximum of 7 per operation.
Arrays and tables	Maximum of 200 per program.
Array/table input record length for compile time	Maximum length is 80.
Control fields (position 59 and 60 of input specifications) length	Maximum length is 256.
Data structure length	Maximum of 9999.
Data structure occurrences (number of)	Maximum of 9999 per data structure.
Do-groups (nested)	Maximum of 100 per program.
Elements in an array/table (positions 36 through 39 of extension specifications)	Maximum of 9999 per array/table.
File	Maximum of 50 per program.
Look-ahead	Can be specified only once for a file. Can be specified only for primary and secondary files.
Overflow indicator	Only one unique overflow indicator can be specified per printer file.
Primary file (P in position 16 of file description specifications)	Maximum of one per program.
Printer file (PRINTER in positions 40 through 46 of file description specifications)	Maximum of eight per program.
Printing lines per page	Minimum of 18; maximum of 112.
Program status data structure	Only one allowed per program.
Record address file (R in position 16 of file description specifications)	Only one allowed per program.
Record length for program described file (positions 24 through 27 or file description specifications)	Maximum length is 9999.[1]
Subroutines	Maximum of 254 per program.
Tables (see arrays)	

[1] Any device record size restraints override this value.

Appendix F. Distributed Data Management (DDM) Files

Distributed Data Management (DDM) allows you to access data files that reside on remote systems with a communications network that supports DDM. The RPG/400 compiler supports DDM files: you can retrieve, add, update or delete data records in a file that resides on another system.

For more information about accessing remote files, refer to the *Communications: Distributed Data Management User's Guide.*

Appendix G. Converting from System/38 to AS/400 System

Features of System/38 Not Supported on AS/400 System

- Card devices
- Mixed device, BSC, and CMN files only supported in System/38 Environment.

Features of AS/400 System That Will Not Run on System/38

- System/38 does not support the AS/400 naming convention for file names. The AS/400 object naming convention is library-name/object-name. The RPG/400 compiler will accept the System/38 naming convention (OBJECT.LIBRARY) only when the compiler is called under the System/38 Environment. Any run time messages issued by the System/38 Environment will display the library, file, member and program names in the AS/400 naming convention (LIBRARY/OBJECT). The /COPY statement and the CALL and FREE operation codes require AS/400 naming convention when used in native and the System/38 naming convention when used in System/38 Environment.
- RPG/400 supports ICF files in AS/400 System native environment.
- The RPG/400 POST feedback area has increased to cover bytes 241 to 528 in the native environment.
- AS/400 System supports numeric variables up to and including 30 digits. The maximum number of decimal digits allowed remains 9.
- AS/400 System supports numerics for the operation code MOVEA.
- AS/400 System has three built-in subroutines: SUBR23R3, SUBR40R3, and SUBR41R3 that are not available on System/38.
- AS/400 System has additional *STATUS values in the native environment.
- AS/400 System places a limit of three on the number of parameter values that you may enter on the command line without specifying the parameter keyword in the native environment.

Converting to AS/400 System

The following table presents the differences between the compiler options for System/38 RPG III and AS/400 RPG/400. Each option is a parameter value for the parameter keyword that immediately precedes it.

RPG/400 Parameter	System/38 Parameter	RPG/400 Options	System/38 Options	Comments
REPLACE	N/A			new parameter
		*YES	N/A	new option

RPG/400 Parameter	System/38 Parameter	RPG/400 Options	System/38 Options	Comments
		*NO	N/A	new option
AUT	PUBAUT			AUT replaces PUBAUT
		*CHANGE	*NORMAL	*CHANGE replaces *NORMAL
		*EXCLUDE	*NONE	*EXCLUDE replaces *NONE
		*USE	N/A	new option
		authori-zation- list-name	N/A	new option
PGM	PGM			Existing Parameter
		*CURLIB	N/A	new option/new default
SRCFILE	SRCFILE			Existing Parameter
		*CURLIB	N/A	new option
PRTFILE	PRTFILE			Existing Parameter
		*CURLIB	N/A	new option
OPTION	OPTION			Existing Parameter
		*SECLVL	N/A	new option
		*NOSECLVL	N/A	new option

Appendix H. Glossary of Abbreviations

Acronyms	Stands For	Definition
CL	Control Language	The set of all commands with which a user request functions.
OS/400	N/A	The operating system for the AS/400 System. It provides many functions that are fully integrated in the system such as work management, data base, data management, job control, message handling, security, programming aids, and service.
DDS	Data Description Specification	A description of the user's data base for device files that is entered into the system using a fixed-form syntax. The description is then used to create files.
DDM	Distributed Data Management	A function of the operating system that allows an application program or user on a source system to access data files on remote systems connected by a communications network that also uses DDM.
GDDM	Graphical Data Display Manager	A group of routines that allows pictures to be defined and displayed procedurally through function routines that correspond to graphics primitives. Contrast with *presentation graphics routines*.
ICF	Intersystem Communications Function	A feature of OS/400 that allows a program to interactively communicate with another program or system.
DBC	Double Byte Characters (DBC) Variables	Variables that can contain ideographic data (Japanese and Chinese symbols, for example). This data is represented by a series to two-byte characters enclosed by the shift-out (S/O) control character (hexadecimal 0E) and the shift-in control character (hexadecimal 0F).
PGR	Presentation Graphics Routines	A group of routines that allows business charts the be defined and displayed procedurally through function routines. Contrast with *Graphical Data Display Manager*.
RPG	Report Program Generator	A high-level language designed for business applications.

Acronyms	Stands For	Definition
SNA	System Network Architecture	The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of Systems Network Architecture networks. **Note:** The layered structure of SNA allows the ultimate origins and destinations of information (that is, the end users) to be independent of, and unaffected by, the specific SNA network services and facilities used for information exchange.
SQL	Structured Query Language	A language that can be used within programs written in other languages, or interactively to access data base manager data and to control access to data base manager resources.
CPF	The system support licensed program for System/38.	It provides many functions that are fully integrated in the system such as work management, data base, data management, job control, message handling, security, programming aids, and service. The equivalent facility in the AS/400 System is OS/400.

Appendix I. Using Double Byte Character Set (DBCS) Data in RPG/400 Programs.

This appendix describes only those enhancements made to RPG/400 for writing programs that process DBCS data. It describes where to use DBCS data in an RPG/400 program, and considerations for working with DBCS date in RPG/400. For additional information on using DBCS see "Moving Double-byte Data and Deleting Control Characters (SUBR40R3)" on page 11-8 and "Moving Double-byte Data and Adding Control Characters (SUBR41R3)" on page 11-10.

Where You Can Use DBCS Data in RPG/400 Programs

In an RPG/400 program, you can use DBCS data:

- In any comment statement (a statement with an asterisk in position 7)
- In the comment field of the extension specification (positions 58-74)
- In the comment field of the calculation specification (positions 60-74)
- In literals and constants
- As data in any character field, subfield, array, or table.

You may mix DBCS data with alphanumeric data. The RPG/400 compiler does not check that each shift-out character is matched with a shift-in character, or that you have entered only DBCS data between these control characters.

RPG/400 uses the value 10 in the *OUT field in the file information data structure (INFDS) to indicate that the national language output capability of the device makes it possible for you to use DBCS data with it.

How to Work with DBCS Data in RPG/400 Programs

Transparent Literals and Constants

A constant or literal is transparent if the constant or literal:

- Begins with an apostrophe followed by a shift-out character
- Ends with a shift-in character followed by an apostrophe
- Contains no embedded shift control characters
- Contains only DBCS data

If you use DBCS data in a constant or literal that is not transparent, you may have an unwanted result. The DBCS data may include a character representing an apostrophe. This character ends your constant or literal where you did not expect it to end. With a transparent constant or literal, RPG/400 makes sure this result does not occur.

If you specify 1 in position 57 of the control specifications, RPG/400 scans for transparent literals and constants. When RPG/400 finds an apostrophe followed by a shift-out character, it checks that your literal or constant is transparent. If it is not, RPG/400 issues a warning message on the compiler listing. The literal or constant is then treated as a literal or constant that is not transparent.

All RPG/400 restrictions on the length of constants or literals apply to transparent constants and literals, including the apostrophes and control characters.

You may use transparent constants and literals in any of the places you use constants or literals:

- In factor 1 and factor 2 of the calculation specifications. However, it may not be meaningful to use DBCS data in all of these places. For example, a move zone operation with a transparent literal in factor 2 will move only the zone of the shift control character. See the description of each operation code in the *Languages: RPG/400 Reference* to decide if it is appropriate to use a transparent literal.
- As a constant in the the constant or edit word field (positions 45-70) of the output specifications.

Additional Considerations for Using DBCS Data in RPG/400 Programs

Here is some information for you to consider when using DBCS data in RPG/400 programs:

- Fields with DBCS data from externally defined files are defined by RPG/400 as character fields. RPG/400 treats these fields in the same manner as it treats other character fields. However, on the cross-reference listing, the DBCS fields are indicated by IGC in the key field section, or by G in the field section.

Example of Coding DBCS Data in an RPG/400 Program

Here are some RPG/400 specifications that include DBCS data.

In this figure, 'Kn' represents a DBCS character, 'o' represents shift-out character, and 'i' represents a shift-in character.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
H                                                 1
H
```

You may check transparent literals and constants by specifying a 1 in column 57 of the control specifications.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
 *
 *oK1K2K3K4iABoK1K2K3K4iCDEFGHoK1K2K3K4K5K6K7K8K9i

*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
E                   ARR         4  5 0             oK1K2i ARRoK1i
E*
```

DBCS data may be mixed with alphanumeric data in comments.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
C           'oK1K2K3i'COMP FLDA                161718
C                     MOVE 'oK1K2K3i'FLDB
C           LABELA    TAG                            'oK1K2K3K4i'
C*
```

Transparent constants and literals are DBCS data enclosed in apostrophes.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...*
0                                   7 'oK1K2i'
0                                  48 'oK1K2K3K4K5K6K7i'
0                         FLDC     57 '$  0. oK1K2K3i'
```

You may use DBCS constants in your output specifications.

Here are some sample entries from a compile listing. FLDK1 is a DBCS field in an externally described file.

```
                    PHYSICAL      LOGICAL
      FILE/RCD      FIELD         FIELD      ATTRIBUTES


                    FLDK1                    IGC      6
                    AFLD                     CHAR     3
```

Figure I-1. Sample Key Field Information

```
      FIELD       ATTR       REFERENCE (M=MODIFIED D=DEFINED)


      FLDK1       G(6)        1000010D
      AFLD        A(2)        1000004D
```

Figure I-2. Sample Cross Reference Listing

Glossary of Terms

access path. The means by which OS/400 provides a logical organization to the data in a data base file so that the data can be processed by a program. See also *arrival sequence access path* and *keyed sequence access path.*

accumulating. The process of totaling the value of a particular field as records are being processed.

alphabetic character. Any one of the letters A through Z (uppercase and lowercase) or one of the characters #, $, or @.

alphameric character. Any one of the alphabetic characters, one of the digits 0 through 9, or the character _ (underscore) as defined in OS/400.

alternate collating sequence. A collating sequence that differs from the normal collating sequence or that allows two or more characters to be considered equal. See *collating sequence.*

alternating array or table. Two arrays or two tables that are loaded together.

AND relationship. The specification of conditioning indicators so that the operation is processed only when all conditions are met.

application program. A program used to do a particular data processing task such as inventory control or payroll.

arithmetic operation. An operation such as addition, subtraction, multiplication, or division that is processed only on numeric fields.

array. A series of elements with like characteristics. An array can be searched for a uniquely identified element, or elements in an array can be accessed by their position relative to other elements. Contrast with *table.*

array element. A data item in an array.

array file. An input file containing array elements.

array index. The actual number of an element in an array, or the field containing the number or relative position of an element in an array.

arrival sequence access path. An access path that is based on the order in which records are stored in a physical file. Contrast with *keyed sequence access path.*

ascending sequence. The arrangement of data in an order from low to high based on the contents of a specific field or fields. Contrast with *descending sequence.*

auto report. A function of the RPG/400 licensed program that uses simplified specifications and standard RPG specifications to generate a complete RPG source program.

auto report option specifications. An RPG coding form the programmer uses to specify options for an auto report program.

auto report program. A set of instructions (program) that use the RPG auto report function. See also *auto report.*

batch processing. A method of running a program or a series of programs in which one or more records (a batch) is processed with little or no interaction with the user or operator. Contrast with *interactive processing.*

binary. Relating to, being, or belonging to a numbering system with a base of 2. Valid digits are 0 and 1.

binary format. Representation of a decimal value in which each field must be 2 or 4 bytes long. The sign (+ or -) is in the leftmost bit of the field, and the integer value is in the remaining bits of the field. Positive numbers have a 0 in the sign bit and are in true form. Negative numbers have a 1 in the sign bit and are in twos complement form.

binary synchronous communications. A form of communications line control that uses transmission control characters to control the transfer of data over a communications line. Abbreviated BSC.

blank after. An output specification that changes the contents of a field so that it contains only zeros (if it is a numeric field) or blanks (if it is a character field) after that field has been written to the output record.

branching. The technique of bypassing specific instructions or operations to alter the sequential running of instructions in a program.

breakpoint. A place in a program (specified by a command or a condition) where the system stops the program and gives control to the work station user or to a specified program.

browse. In SEU, to look at records in a source member or in a spooled output file by using the record-positioning operations, the scan operation, and the roll keys.

byte. A group of eight adjacent binary digits that represents one EBCDIC character.

calculation specifications. An RPG coding form on which the programmer describes the processing to be done by the program.

called program. A program whose processing is requested by another program (a calling program) or by a command.

calling program. A program that requests the processing of another program (a called program).

character. Any letter, digit, or other symbol in the data character set that is part of the organization, control, or representation of data.

character field. An area that is reserved for a particular unit of information and that can contain any of the characters in the data character set. Contrast with *numeric field.*

character literal. A symbol, quantity, or constant in a source program that is itself data, instead of a reference to data. Contrast with *numeric literal.*

collating sequence. The order each character holds in relation to other characters according to the bit structure.

combined file. A data file that is used as both an input file and an output file. The output file contains only those fields described for the output file (that is, the output record does not necessarily contain the same fields as the input record).

comment. A word or statement in a program, command, or file that serves as documentation instead of as instructions. A comment is ignored by a compiler.

compilation. Translation of a source program (such as RPG/400 specifications) into a program that can be run.

compile. To translate a source program into a program that can be run (an object).

compile-time array or table. An array or table in which the data is compiled with the source program and becomes a permanent part of the program. Contrast with *run-time array* and *prerun-time array or table.*

compiler. A program that translates a source program into a program that can be run.

compiler-directive statement. An instruction that causes the compiler to take a specific action during compile time. Directive statements generally occur in compiler source input.

compiler /COPY. A compiler directive similar in function to the auto report /COPY. The compiler /COPY, however, is limited to just the inclusion of copied records (members). Other functions, such as specification sorting and replacement, are not supported.

compiler listing. A printout that is produced by compiling a program or creating a file and that optionally includes, for example, a line-by-line source listing, a cross-reference list, diagnostic information, and for programs, the description of externally described files.

composite key. A key for a file or record format that is composed of more than one key field.

conditioning. In a program, the use of indicators to control when certain functions or operations are to be done. For example, in an RPG program indicators can control calculation or output operations.

consecutive processing. A method of file processing in which records are read in the order in which they exist in the file.

constant. Data that has an unchanging, predefined value to be used in processing. A constant does not change during the running of a program, but the contents of a field or variable can. See also *literal.*

continuation lines. Additional lines specified on the file description specifications to provide more information about the file being defined.

control break. A change in the contents of a control field that indicates all records from a particular control group have been read and a new control group is starting.

control field. One or more fields that are compared from record to record to determine when the information in the fields changes. When the information changes, the control level indicator (L1 through L9) assigned to a control field is set on.

control language. The set of all commands with which a user requests functions. Abbreviated CL.

control language program. An object that is created from source consisting entirely of control language commands.

control level indicator. An indicator (L1 through L9) used to identify certain fields as control fields and then

used to condition which operations are to be processed at detail or total calculation or output time.

control specification. An RPG coding form on which the programmer provides information that affects program generation and running

control statement. Entries on a control specification.

create. The function used to bring an object into existence in the system.

cross-reference listing. The portion of the compiler listing that contains information on where files, fields, and indicators are defined, referenced, and modified in a program.

currency symbol. A character such as the dollar sign ($) used to identify monetary values.

data area. An object that is used to communicate data such as CL variable values between the programs within a job and between jobs. The system-recognized identifier for the object type is *DTAARA.

data area data structure. A data structure that is used to retrieve data from a data area.

data base. The collection of all data base files stored in the system.

data base file. An object that contains descriptions of how input data is to be presented to a program from internal storage and how output data is to be presented to internal storage from a program. See also *physical file* and *logical file.*

data character set. The 256 EBCDIC characters.

data communications. The transmission of data between systems and/or remote devices over a communications line.

data description specifications. A description of the user's data base or device files that is entered into the system using a fixed-form syntax. The description is then used to create files. Abbreviated DDS.

data structure. An area in storage that is composed of one or more subfields. A data structure can be either program-described or externally described.

default error handler. The portion of the RPG logic cycle that handles program or file exception/errors when program or file exception/errors are not controlled by the programmer.

descending sequence. The arrangement of data in an order from high to low based on the contents of a specific field. Contrast with *ascending sequence.*

detail line. A detail record in an output file.

detail output. Specified output operations that are processed for every record read.

detail record. An output record produced during the detail output operation of the RPG program cycle. Contrast with *total record.*

detail time. That part of the RPG program cycle in which calculation and output operations are processed for each record read. Contrast with *total time.*

diagnostic message. A message that contains information about errors in the processing of an application program or a system function.

Distributed Data Management (DDM). A function of the operating system that allows an application program or user on a source system to access data files on remote systems connected by a communications network that also uses DDM.

do group. Group of operations that can be processed repeatedly and is delimited and controlled by a DO, DOUxx, DOWxx, or IFxx operation and an associated END operation.

dump. To copy data in a readable format from main or auxiliary storage onto an external medium such as tape, diskette, or printer.

dynamic. Occurring at run time.

edit. To modify a numeric field to an external format by suppressing zeros and inserting commas, periods, currency symbols, the sign status, or other constant information.

edit code. A letter or number indicating what kind of editing should be done before a field is displayed or printed.

edit word. A user-defined word with a specific format that indicates how editing should be done.

element. The smallest addressable unit of an array or table.

end position. An entry in the output specifications that indicates where the end position of a field or constant is to be placed in the output record.

exception/error handling. Programmer-written code that handles program or file exceptions/errors. If the

program or file exceptions/errors are not controlled by the programmer, the RPG default error handler receives control.

extension and line counter specifications. An RPG coding form on which the programmer provides information about record address files, arrays, and tables and their associated files used by a program and about the number of lines to be printed on the printer forms that are used.

external indicators. Indicators that can be set by another program prior to running a program or altered while the program is running. Valid external indicators are U1 through U8.

externally described file. A file for which the fields in the records are described to OS/400, through data description specifications, when the file is created. The field descriptions can be used by the program when the file is processed. Contrast with *program-described file.*

factor. An entry (for example, a field name, file name, literal, or data structure) that identifies the data to be used in a calculation operation.

fetch overflow. A routine that allows the user to alter the basic RPG overflow logic to prevent printing over the perforation and to allow the user to use as much of the page as possible.

field. An area that is reserved and used for a particular item of information.

field indicator. An indicator used to indicate whether a given field in an input record is plus, minus, zero, or blank.

field record relation indicator. An indicator used to associate fields in an input record with a particular record type. The field record relation indicator is normally used when the record type is one of several in an OR relationship.

figurative constant. An implied literal that is specified in the calculation specifications without a length definition because the implied length and decimal positions are the same as those of the receiver field.

file. A generic term for the object type that refers to a data base file, a device file, or a set of related records treated as a unit. The system-recognized identifier for the object type is *FILE.

file description specifications. An RPG coding form on which the programmer identifies and describes all files used in a program.

file exception/error subroutine. A user-written subroutine that may receive control following file exceptions/errors.

file information data structure (INFDS). A data structure that can be defined for each file to make file exception/error information available to the program. A file information data structure must be unique for each file.

file key. All the key fields defined for a file.

file operation code. An operation code (for example, CHAIN) that lets the user control the input/output operations to a file.

file translation. A function that is used to change any of the 256 EBCDIC characters into another EBCDIC character.

first-page indicator. An indicator used to specify which records (such as headings) should be written on the first cycle only. Abbreviated 1P.

fixed currency symbol. A currency symbol that appears in a specified position of an edited field. Contrast with *floating currency symbol.*

floating currency symbol. A currency symbol that appears immediately to the left of the leftmost position in an edited field. Contrast with *fixed currency symbol.*

full procedural file. A file for which the input operations are controlled by programmer-specified operation codes instead of by the program cycle. Contrast with *primary file.*

Function key indicator. An indicator that is set on when a system operator or work station user presses a valid corresponding function key. Valid function key indicators are KA through KN and KP through KY.

graphical data display manager. A group of routines that allows pictures to be defined and displayed procedurally through function routines that correspond to graphics primitives. Abbreviated GDDM. Contrast with *presentation graphics routines.*

graphics. (1) The making of charts and pictures. (2) Pertaining to charts, tables, and their creation.

half adjust. A method of rounding off a number by adjusting the last digit to be kept. When the number to the right of the last digit to be retained is 5 or greater, 1 is added to the last retained digit. For example, 2.475 half-adjusted to two decimal places becomes 2.48, but 2.474 becomes 2.47.

halt indicators. An indicator used to end the program when an unacceptable condition occurs. Valid halt indicators are H1 through H9.

heading. A constant, or field, usually at the top of a page or display, that identifies the information on the page or display.

hexadecimal. Pertaining to a numbering system with a base of 16. Valid numbers are the digits 0 through 9 and the characters A through F, where A represents 10 and F represents 15.

indexed file. A data base file whose access path is built on key values. Each record in the file is identified by a key field.

indicator. (1) A 2-character entry on a specification form that is used to test a field or record or to tell when certain operations are to be processed. (2) An internal switch used by a program to remember when a certain event occurs and what to do when the event occurs.

input. Information (or data) to be processed.

input file. A data base or device file that has been opened with the option to allow records to be read.

input specifications. An RPG coding form on which the programmer describes the records and their fields in a program-described input file, adds RPG functions to an externally described input file, or defines a data structure and its subfields.

instruction. A statement that specifies an operation to be processed by the system and that identifies the data, if any, involved in that operation.

integer. Any of the natural numbers, the negatives of these numbers, or zero.

interactive processing. Pertaining to a program or procedure that alternately accepts input and then responds to the input. Contrast with *batch processing*.

Intersystem Communications Function (ICF). A feature of OS/400 that allows a program to interactively communicate with another program or system.

key field. A field in a record whose contents are used to sequence the records of a particular type within a file member.

keyed sequence access path. An access path to a data base file that is ordered on the contents of key fields contained in the individual records. Contrast with *arrival sequence access path.*

label. A symbolic name that represents a specific location in a program. A label can serve as the destination point for one or more branching operations.

last record indicator. An indicator that signals when the last data record is processed. This indicator can then be used to condition calculation and output operations that are to be done at end of program. Abbreviated LR.

left-adjust. To place an entry in a field or to move the contents of a field so that the leftmost character of the data is in the leftmost position of the field.

level zero entry. An entry on calculation specifications indicating operations that are to be done during total time for each program cycle when no control break has occurred. Abbreviated L0.

library. An object that serves as a directory to other objects. A library is used to group related objects and to find objects by name when they are used. The system-recognized identifier for the object type is *LIB.

library list. An ordered list of library names used to find an object. The library list indicates which libraries are to be searched and the order in which they are to be searched. The system-recognized identifier is *LIBL. *LIBL specifies to the system that a job's current library list is to be used to find the object.

limits file. A record address file containing limits records when the sequential-within-limits processing method is used.

limits record. A record that consists of the lowest record key and the highest record key of the records in the keyed file that are to be read.

line counter specifications. An RPG coding form on which the programmer indicates or overrides the system defaults for the length of the printer form and the number of lines to print on a page. Line counter specifications can be used for each printer file in a program.

literal. A character string whose value is given by the characters themselves. For example, the numeric literal 7 has the value 7, and the character literal 'CHARACTERS' has the value CHARACTERS. See also *character literal, constant*, and *numeric literal.*

logical file. A description of how data is to be presented to or received from a program. This type of data base file contains no data, but it provides an ordering and format for one or more physical files. Contrast with *physical file.*

look-ahead field. A field that allows the program to look at information in a field on the next record that is available for processing in an input or update file.

match fields. In primary or secondary multifile processing, fields within a record type that are to be used for sequence checking of a single file, or for matching records of one file with those of another file.

match level. The value (M1 through M9) assigned to the match field. The match level identifies fields by which records are matched during primary or secondary multifile processing.

matching record indicator. An indicator used in calculation or output specifications to indicate operations that are to be processed only when records match in primary and secondary files. Abbreviated MR.

message file. An object that contains message descriptions. The system-recognized identifier for the object type is *MSGF.

modular program design. A design in which multiple programs perform functions (normally one program per function). Modular program design applies to both batch and interactive processing.

multiple device file (MDF). In RPG, any WORKSTN file with one of the keywords ID, IND, NUM or SAVDS. Such a file can access more than one device, and devices of various types.

multiple occurrence data structure. A data structure that has more than one copy of the data structure contained in a program.

nested do group. A do group that is contained within another do group.

numeric character. Any one of the digits 0 through 9.

numeric field. An area that is reserved for a particular unit of information and that can contain only the numeric digits 0 through 9. Contrast with *character field.*

numeric literal. The actual numeric value to be used in processing, instead of the name of a field containing the data. A numeric literal can contain any of the numeric digits 0 through 9, a sign (plus or minus), and a decimal point. Contrast with *character literal.*

object. A named unit that consists of a set of attributes (that describe the object) and, in some cases, data. An object is anything that exists in and occupies space in storage and on which operations can be processed. Some examples of objects are programs, files, and libraries.

operation. A defined action processed on one or more data items, such as adding, multiplying, comparing, or moving information.

operation code. A word or abbreviation, specified in the calculation specifications, that identifies an operation.

OR relationship. The specification of conditioning indicators such that the conditioned operation is done when any one of the conditions is met.

output. Data transferred from storage to an output device.

output file. A data base or device file that has been opened with the option to allow records to be written.

output indicator. An indicator used to define the conditions under which an output record or an output field in the output specifications is written. An output indicator must be previously defined before it is used in the output specifications.

output specifications. An RPG coding form on which the programmer describes the records and their fields in a program-described output file or adds RPG functions to an externally described output file.

overflow. The condition that occurs when the last line specified as the overflow line to be printed on a page has been passed.

overflow indicator. An indicator that signals when the overflow line on a page has been printed or passed. The indicator can be used to specify which lines are to be printed on the next page.

overflow line. The line specified as the last line to be printed on a page.

overflow page. The new page created when overflow occurs.

packed decimal format. Representation of a decimal value in which each byte within a field represents two numeric digits except the rightmost byte, which contains one digit in bits 0 through 3 and the sign in bits 4 through 7. For all other bytes, bits 0 through 3 represent one digit; bits 4 through 7 represent one digit. For example, the decimal value +123 is represented as 0001 0010 0011 1111. Contrast with *zoned decimal format.*

packed field. A field that contains data in the packed decimal format.

packed key. A key in the packed decimal format.

parameter. Data passed to or received from another program.

parameter list. A list of values that provides a means of associating addressability of data defined in a called

program with data in the calling program. It contains parameter names and the order in which they are to be associated in the calling and called program.

physical file. A description of how data is to be presented to or received from a program and how data is actually stored in the data base. A physical file contains one record format and one or more members. Contrast with *logical file.*

prerun-time array or table. An array or table that is loaded at the same time as the source program, before the program actually begins running. See also *compile-time array or table* and *run-time array.*

presentation graphics routines. A group of routines that allows business charts to be defined and displayed procedurally through function routines. Abbreviated PGR. Contrast with *Graphical Data Display Manager.*

primary file. If specified, the main file from which RPG first reads a record in the program cycle. In multifile processing, the primary file is used to determine whether the MR indicator is set on. Contrast with *full procedural file.*

processing. The action of doing operations on input data.

program cycle. A series of steps done by a compiled RPG program in a specific order for each primary or secondary record that is read.

program-described data. Data contained in a file for which the fields in the records are described in the program that processes the file. Contrast with *externally described data.*

program device. A symbolic mechanism that a program uses instead of a real device (identified by the device name) to access the devices in a file. When the program uses a program device, the system redirects the operation to the appropriate real device.

program message queue. A message queue used to hold messages that are sent between program invocations of a routing step. The program message queue is part of the job message queue.

prompt. A displayed requested for information or user action. The user must respond to allow the program to proceed.

random-by-key processing. A method of processing a full procedural file by using the CHAIN operation code. Records to be processed are identified by record keys.

random-by-relative-number processing. A method of processing a full procedural file by using the CHAIN operation codes. Records to be processed are identified by relative record numbers.

read-from-invited-program-devices operation. In OS/400, an input operation that waits for input from any one of the invited program devices for a user-specified time. Contrast with *read-from-one-program-device operation.*

read-from-one-program-device operation. In OS/400, an input operation that will not complete until the specified device has responded with input. Contrast with *read-from-invited-program-devices operation.*

record. An ordered set of fields that make up a single occurrence of the basic unit of data transferred between a file and a program.

record address file. An input file that indicates which records are to be read from another file, and the order in which the records are to be read from the file.

record format. The definition of how data is structured in the records contained in a file. The definition includes the record name, field names, and field descriptions (such as length and data type). The record formats used in a file are contained in the file's description.

record identification code. Characters placed in a record to identify that record type.

record identifying indicator. An indicator that identifies the record just read.

record key. All the key fields defined for the record type.

record type. The classification of records in a file. Records of the same type have the same fields in the same order. For program-described files, these records have record identification codes; for externally described files, the records have the same record format name.

relational checking. Diagnostics done against two statements in a source program to ensure that the statements are valid (for example, a GOTO operation must have an associated TAG operation). This type of checking is done only by the compiler as opposed to single-statement syntax checking that is done by the SEU syntax checker.

relative end position. An entry on the output specifications that indicates the number of blank positions that are to appear between a field or constant and the field or constant defined on the preceding specification line. Contrast with *exact end position.*

relative record number. A number that specifies the location of a record in relation to the beginning of a

data base file member or subfile. For example, the first record in a data base file member or subfile has a relative record number of 1.

reserved word. (1) A word in the RPG language (such as *PLACE or UDATE) that is associated with a specific language function. A reserved word cannot be used as a programmer-defined symbolic name. (2) A special word that has a specific meaning to the system as defined in a programming language.

result field. A field specified on a calculation operation that contains the result of that operation.

resulting indicator. An indicator that signals the result of a calculation, such as whether the result is plus, minus, or zero; whether a given field is greater than, less than, or equal to another field; or whether an operation was successfully completed.

return indicator. An indicator used to indicate to the internal RPG logic that control should be returned to the calling program. Abbreviated RT.

right-adjust. To place an entry in a field or to move the contents of a field so that the rightmost character of the data is in the rightmost position of the field.

running. The carrying out of the instructions of a computer program by a processing unit.

run-time array. An array that is loaded or created by input or calculation specifications after the program starts running. Contrast with *compile-time array or table* and *prerun-time array or table.*

search argument. (1) A literal or field name specified in factor 1 of certain file operations (such as CHAIN) that identifies the record to be processed. (2) Data for which you want to find a match or a greater than or less than quantity in a table or array. The search argument is specified in the lookup statement.

secondary file. In multifile logic, any input file other than the primary file.

sequence checking. An RPG function that checks the sequence of records in input, update, or combined files used as primary and secondary files.

sequential-by-key processing. A method of file processing that reads records from a keyed sequence file in the order in which the keys are arranged in the access path.

sequential file. A file in which records are processed in the order that they are stored in the file.

sequential-within-limits processing. A method of file processing in which a limits record address file specifies the keys of the records to be read from a keyed file.

single program design. A design in which all functions are done within one program.

source entry utility. The utility of the Interactive Data Base Utilities licensed program that is used to create and change source members. Abbreviated SEU.

source file. A file created by the specification of FILETYPE(*SRC). A source file can contain source statements for such items as high-level language programs and data description specifications.

source listing. A portion of a compiler listing that contains source statements and diagnostics. See also *compiler listing.*

source program. A set of instructions, written in a programming language such as RPG or COBOL, that represents a particular job as defined by a programmer. A source program is used as input to the compiler to create a program that can be run.

special character. A character other than a digit, a letter, or #, $, or @. For example, *, +, and % are special characters.

subfield. A field within a data structure.

subfile. A group of records of the same record format that can be displayed concurrently as a work station. The system sends the entire group of records to the work station in a single operation and receives the group in another operation.

subroutine. A group of calculation specification statements in a program that can be run several times in that program.

symbolic name. A unique name used to define an entity such as a field, file, data structure, or label within an RPG program.

syntax checking. A function of the command analyzer, a compiler, or SEU that checks single statements for violations of the rules governing the structure of the statement.

Systems Network Architecture. The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of Systems Network Architecture networks. Abbreviated SNA. Note: The layered structure of SNA allows the ultimate origins and destinations of information (that is, the end users) to be independent of, and unaffected by, the spe-

cific SNA network services and facilities used for information exchange.

table. A series of elements with like characteristics. A table can be searched for a uniquely identified element, but elements in a table cannot be accessed by their position relative to other elements. Contrast with *array*.

table file. An input file that contains a table.

total operations. Operations (calculation or output) processed only after a control group of records has been processed or at end of program.

total record. An output record written after a group of detail records. Total records generally contain data that is the result of calculations processed on the information in a group of detail records. Contrast with *detail record*.

total time. That part of the RPG program cycle in which calculation or output operations specified for a group of records are done. Total time operations are conditioned by control level indicators (L1 through L9 or LR). Contrast with *detail time*.

update file. A file that is used as both an input file and an output file. If the program alters the data in one or more fields of an input record and then writes the altered fields and the unaltered fields back to the same positions in the same record, the file is an update file.

user-defined edit code. A number (5 through 9) indicating that editing should be done on a numeric output field according to a pattern predefined to OS/400. User-defined edit codes can take the place of edit words, so that repetitive coding of the same edit word is not necessary.

validity checker. A user-written program that tests commands for errors in the parameter values. Validity checking is done in addition to the checking done by the command analyzer.

zero suppression. The substitution of blanks for leading zeros in a number. For example, 00057, when zero suppressed, becomes VVV57 (V represents one blank space).

zoned decimal format. Representation of a decimal value by 1 byte per digit. Bits 0 through 3 of the rightmost byte represent the sign; bits 0 through 3 of all other bytes represent the zone portion; bits 4 through 7 of all bytes represent the numeric portion. For example, in zoned decimal format, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Contrast with *packed decimal format*.

zoned field. A field that contains data in the zoned decimal format.

Index

A

access path
  definition GLOSS-1
  example of 8-10
  for externally described DISK file
    arrival sequence 8-2
    keyed sequence 8-2
  for indexed file 8-8
adding RPG function to external description 6-10
alphabetic characters, definition GLOSS-1
alphameric
  definition GLOSS-1
  field
    definition GLOSS-1
  literal
    definition GLOSS-1
alternate collating sequence
  coding form 15-1
  control specification entry 15-1
  definition GLOSS-1
  input record format 15-4
  operations affected 15-1
alternating format (arrays and tables)
  definition GLOSS-1
  example 13-9
ALTSEQ 15-1
ampersand (&)
  auto report copy function 16-7
  in body of edit word 14-14
  in status of edit word 14-10
AND relationship
  definition GLOSS-1
apostrophe
  use with edit word 14-14
application design
  modular program 1-19
  single program 1-18
area parameter for SPECIAL PLIST 6-30
arithmetic operation codes
  See calculation operations
  See factor 1
  See factor 2
  See half-adjust
  See individual entries
arithmetic operation codes *(continued)*
  See operation codes
array
  adding entries to an array 13-16
  alternating
    definition GLOSS-1
    examples 13-9
  auto report specifications
  binary format 10-3
  building
    See array, loading
  building via calculations
    See run-time array
  calculating totals with arrays 13-18
  combined array file 13-4
  compile-time
    arrangement in source program 13-8
    definition of 13-4, GLOSS-2
    rules for loading 13-4
  creating input records 13-5
  differences from table 13-1
  dynamic arrays
    See run-time array
  editing 13-18
  element 13-18
    definition GLOSS-3
  entries
  examples of using 13-18
  extension specifications
  file
  index 13-2
    field 13-19
    fixed 13-20
  loading
    compile-time 13-4
    from more than one record 13-3
    from one record 13-3
    prerun-time 13-8
    run-time 13-3
    specifications (see run-time arrays)
    via input or calculation specifications (see run-time arrays)
  modifying contents 13-16
  name
    and index 13-2

array *(continued)*
name *(continued)*
how to form 13-2
invalid 13-2
rules for 13-7
valid 13-2
order in source program 13-8
output 13-17
formatting 13-20
printing elements 13-25
packed format 10-1
prerun time
definition of 13-7, GLOSS-7
rules for loading 13-8
printing elements of 13-25
referring to in calculations 13-15
run-time
definition of 13-2, GLOSS-8
rules for loading 13-3
with consecutive elements 13-4
with scattered elements 13-5
searching
an array with an index 13-13
an array without an index 13-10
using field indexes 13-18
using fixed indexes 13-20
using name and index 13-2
using name only 13-2
arrival sequence access path 8-2
ascending sequence
definition GLOSS-1
asterisk fill (asterisk protection)
in body of edit word 14-3
with combination edit codes 14-3
asterisk (*)
generated specifications 16-1, 16-16
indication on auto report total lines 16-11
AUT parameter 3-12
auto report copy function
See /COPY statement specifications
auto report function
See also *AUTO output specifications
See also /COPY statement specifications
See also ?
A$$SUM subroutine 16-16
compilation of source program 3-1
definition of GLOSS-1
examples of 16-29
auto report function *(continued)*
generated specifications
calculation 16-16
output 16-17
programming aids 16-22
auto report program
data areas A-27
overview A-27
phases A-27
A$$SUM subroutine (auto report) 16-16

B

batch processing, definition GLOSS-1
binary field
comparison with packed and zoned decimal fields 10-3
data structure subfield specifications 10-7
for array/table file 10-3
input specifications 10-3
output specifications 10-3
binary format
array/table field 10-4
definition 10-4, GLOSS-1
description 10-4
input field 10-4
output field 10-4
Binary Synchronous Communication D-1
blank after
definition GLOSS-1
Blocking output records 6-8
body (of an edit word) 14-10
breakpoint
definition GLOSS-1
examples 4-7
general description 4-6
BSC data communication D-1

C

calculation
generated by auto report 16-16
indicators
AND/OR relationship 12-23
conditioning 12-23
control level 12-22
resulting 12-14
specifications
subroutines

calculation specifications 2-2
CALL 'GDDM' 11-5
call a program (CALL) operation code
CALL operation code
  listing A-23
  program communication 11-3
calling other programs 11-1
character
  collating sequence 15-2
  definition GLOSS-2
  hexadecimal equivalents 15-2
checking sequence
  See sequence checking
checking, level 6-7
CL programs
  calling 11-1
  definition H-2
clear command 9-5
code
  See edit codes
CODELIST parameter 3-13, A-7
codes, operation
  See operation codes
coding form
  See calculation specifications
  See control specifications
  See data description specifications
  See extension specifications
  See file description specifications
  See input specifications
  See line counter specifications
  See output specifications
  See translation table and alternate collating sequence coding sheet
collating sequence
  See also alternate collating sequence
  alternate 15-1
  definition GLOSS-2
  normal 15-1
combination edit codes (1-4, A-D, J-M) 14-2
combined file
  definition GLOSS-2
command attention (CA) keys
  See function keys
command function (CF) keys
  See function keys
command keys 9-5
commands
  CRTRPGPGM 3-1
  CRTRPTPGM 16-26
comments
  on array input records 13-5, 13-8
  /COPY statement 16-4
commitment control 7-1
common area
  VCOMMON A-4
  XREGN A-27
communication
  accessing other programs and systems 9-1
  data D-1
  error recovery D-2
  exception/error handling D-1
  with objects in system 11-1
compare, arithmetic
  alternate collating sequence 15-1
compilation
  auto report source program 16-26
  RPG source program 3-1
compile-time array or table
  See also array
  definition GLOSS-2
  general description 13-4
compiler
  data areas A-4
  debugging options A-5, A-7
  directives
    /COPY C-3, GLOSS-2
  error message organization A-4
  overview A-1
  phases A-2
  service information A-1
compiling
  a source program 3-1
  an auto report source program 16-26
composite key
  See search argument
conditional branching 1-10
conditioning files 12-17
  See also external indicators
conditioning indicators
  calculation
    general description 12-22
    positions 7 and 8 12-22
    positions 9 through 17 12-23

conditioning indicators *(continued)*
file
general description 12-17
rules for 12-17
general description 12-17
output
general information 12-17
conditioning output 6-19
explanation of 12-27
consecutive processing 8-15
constants
See also edit word
See also literals
definition GLOSS-2
constants, figurative
See *ALL'X..',
See *BLANK/*BLANKS
See *HIVAL
See *LOVAL
See *ZERO/*ZEROS
control break
See also control field
See also control group
See also control level (L1-L9) indicators
definition GLOSS-2
general description 12-4
how to avoid unwanted 12-5
on first cycle 12-4
control field
See also control break
See also control level (L1-L9) indicators
definition GLOSS-2
general information 12-5
split 12-10
control group
See also control break
See also control field
See also control level (L1-L9) indicators
general information 12-4
control language (CL) program
calling 11-1
definition GLOSS-2
control level (L1-L9) indicators
See also conditioning calculations
See also control break
See also control field
See also control group
as field record relation indicator 12-18
control level (L1-L9) indicators *(continued)*
definition GLOSS-2
examples 12-6, 12-11
general description 12-4
rules for 12-5
setting of 12-35
control record format, subfile 9-7
control specification 2-1
controlled loop 1-11
copy function, auto report
See /COPY statement specifications
CR (negative balance symbol)
with combination edit code 14-2
with edit words 14-13
create, definition GLOSS-3
cross-reference listing
how to request for RPG source program 3-9, A-6
how to request for RPG source program containing auto report specifications 16-27
CRTRPGPGM (create RPG program) command
debugging parameters A-5
description 3-1
CRTRPTPGM (create auto report program) command
description 16-26
currency symbol
See also fixed currency symbol
See also floating currency symbol
use in edit word 14-12
cycle, program
detailed description 5-3
fetch overflow logic 5-11, 6-23
general description 1-5, 1-8
with lookahead 5-11
with match fields 5-10
with RPB exception/error handling 5-8
flowchart 4-37
with RPG exception/error handling 5-11

D

D-*AUTO
See also *AUTO output specifications
overflow of print lines 16-14
data area data structure
definition of GLOSS-3
general information 10-8
statement

data area data structure *(continued)*
statement *(continued)*
externally described 10-6
program described 10-6
rules 10-8
specifications 10-8
subfields
rules 10-12
specifications 10-11
used to access data area 11-13
data areas
compiler A-4
definition GLOSS-3
general information 11-13
how to access 11-13
printing A-6
restrictions 11-13
retrieval
implicit 5-2, 10-9
RETURNCODE 3-2
unlocking
implicit 5-3, 10-9
writing
implicit 5-3, 10-9
data base
file
See also DISK file
definition GLOSS-3
data communication D-1
data description specifications (DDS)
definition GLOSS-3
example of 6-11
data format
See binary format
See packed decimal format
See zoned decimal format
data management feedback area
See also file information data structure
description 4-23
data structure
definition GLOSS-3
examples of 10-13
externally described 10-20
format 10-7
general information 10-6
length of 10-8
multiple occurrences of 10-8
special types
data area 10-9
data structure *(continued)*
special types *(continued)*
file information (INFDS) 4-39, 10-9
program status 4-50, 10-9
statement
externally described 10-6
program described 10-6
rules 10-9
specifications 10-8
subfields
format 10-7
rules 10-12
specifications 10-11
date field
effect of end position 14-5
zero suppression 14-2
date, user
See UDATE
See UDAY
See UMONTH
See UYEAR
DDM
See distributed data management (DDM)
DDS
See data description specifications (DDS)
DEBUG operation code
debugging options
compiler A-5
examples A-7
debugging RPG programs
See breakpoint trace
See DEBUG and DUMP operation codes
decimal data format
See packed decimal format
See zoned decimal format
default error handler, RPG 4-35
defining an alternate collating sequence 15-1
definition of terms GLOSS-1
descending sequence
definition GLOSS-3
designing your program 1-8
designing the input 1-9
designing the output 1-8
designing the processing 1-9
design, application
See application design

detail lines
See *AUTO output specifications
detail printing, auto report
See *AUTO output specifications
detailed program logic 5-3
device files
multiple 6-30
device name, function of 6-2
devices
DISK 8-1
PRINTER 6-17
SEQ 6-27
SPECIAL 6-29
WORKSTN 9-1
DISK file
externally described 8-1
access path 8-2
as program described file 8-3
examples 8-3
processing 8-8
file operation codes allowed
for keyed processing methods 8-30
for nonkeyed processing methods 8-31
processing charts
externally described 8-32
program described 8-34
program described 8-8
as indexed file 8-8
as record address file 8-12
as sequential file 8-9
processing 8-14, 8-15
record format specifications 8-1
summary of processing methods 8-13
display device file
See WORKSTN file
distributed data management (DDM) xv, GLOSS-3
do group 1-17
definition GLOSS-3
Do Until structure 1-14
Do While structure 1-12
double asterisk (**)
generated specifications 16-1, 16-16
Double Byte Characters (DBC) variables 11-11
subroutines SUBR40, SUBR41 11-11
DS
See data structure
dump, formatted 4-13
duplicate field names on /COPY modifier statement 16-9
dynamic array
See run-time array

E

edit codes
combination (1-4, A-D, J-M) 14-2
date field
definition GLOSS-3
effect of inverted print 14-3
effect on end position 14-5
examples 14-15, 14-16
simple (X, Y, Z) 14-2
summary tables 14-2, 14-6
user-defined (5-9) 14-4
zero suppression 14-2
edit source (STRSEU) command 2-3
edit word
definition GLOSS-3
examples 14-15, 14-16
formatting 14-9, 14-14
parts of 14-9
rules for 14-14
editing
date fields 14-2
externally described files 14-15
nonprinter files 14-4
edit, date 14-2
encapsulated program A-1
end of file
with primary file 12-16
end position
effect of edit codes on 14-8
in output record
ending a program 5-11, 11-11
ENDSR (end subroutine) operation code
return points 4-46
enhancements, RPG
See RPG enhancements
entry 13-1
See also array
See also table
error logic 4-37, 5-8
See also exception/error handling

error messages, compiler A-4, D-2
error parameter for SPECIAL PLIST 6-29
error recovery
  communications D-2
exception/error handling
  communications files D-1
  data structures
    INFDS (file information) 4-39
    program status 4-50
  logic flow 4-37, 5-8
  programmer control
    file exception/error subroutine (INFSR) 4-45
    indicator in positions 56 and 57 of calculation specifications 4-39
    program exception/error subroutine 4-52
  RPG default 4-35
  status information
    file exception/error 4-41
    program exception/error 4-50
  subroutines
    INFSR (file exception/error) 4-45
    *PSSR (program exception/error) 4-53
exception/error indicators 4-39
EXCPT name
expansion (of an edit word) 14-10, 14-14
extension specifications 2-2
external (U1-U8) indicators
  as field record relation indicator 12-18
  definition GLOSS-4
  general description 12-15
  resetting 12-15
  setting 12-35
externally described data structure 10-20
externally described file
  adding to external description 6-10
  advantages 6-4
  as program described file 6-4
  as WORKSTN file 9-1
  data format 10-1
  definition GLOSS-4
  editing 14-15
  file description specifications for 6-10
  output specifications for 6-15
  overriding 6-13
  renaming record format 6-10

F

factor 1
factor 2
feedback information in file information
  data structure 4-23
fetch overflow
  See also overflow (OA-OG, OV) indicators
  definition GLOSS-4
  general description 6-21
  logic 5-11, 6-23
field
  control 12-5
  match 6-34
  packed 10-1
field indicators (01-99, H1-H9, U1-U8, RT)
  as halt indicators 12-13
  assigning on input specifications
  general description 12-13
  numeric 12-13
  rules for assigning 12-13
  setting of 12-35
field name
  table
    FLDTBL A-27
    XFDTAB A-4
field record relation indicators (01-99, H1-H9, L1-L9, U1-U8)
  example 12-20
  general description 12-18
  rules for 12-18
field reference file, example of 8-4
file
  array
    See array
  conditioning 12-17
  data base
    See DISK file
  definition GLOSS-4
  deleting records from
    DISK files 8-32, 8-34
    SPECIAL files 6-33
  DISK
    See DISK file
  display device
    See WORKSTN file
  externally described
    See externally described file
  full procedural 5-12

file *(continued)*
general considerations 6-1
indexed 8-8
name
externally described 6-4
override 6-13
program described 6-17
override 6-13
PRINTER
See PRINTER file
processing charts
DISK files 8-32
PRINTER files 6-27
SEQ files 6-29
SPECIAL files 6-33
WORKSTN files 9-15
program described
See program described file
record address
See record address file
redirection 6-2
SEQ
See SEQ file
SPECIAL
See SPECIAL file
table
See table
tape
See SEQ file
WORKSTN
See WORKSTN file
file condition 12-15, 12-17
See also external indicators, UC
file description specifications 2-1
CFILE continuation line option 9-1
definition GLOSS-4
using the copy function with auto report 16-6
file exception/error subroutine (INFSR)
description 4-45
example 4-48
return points 4-46
specifications for 4-45
file exception/errors
examples 4-48
general information 4-38
how to handle
default 4-38
indicators 4-38
subroutine (INFSR) 4-45
file exception/errors *(continued)*
status information
data structure (INFDS) 4-38
file feedback information in file information data structure 4-23
file information data structure
data management feedback area 4-23
feedback information 4-23
general information 4-39, 10-10
predefined subfields 4-21, 4-40
statement
externally described 10-6
program described 10-6
rules 10-8
specifications 10-8
status codes 4-42
subfields
predefined 4-21, 4-39
rules 10-12
specifications 10-9
*FILE 4-41
*OPCODE 4-41
*RECORD 4-42
*ROUTINE 4-42
*STATUS 4-41
file locking by RPG 6-7
file name table, XFLTAB A-4
file operation codes
allowed with DISK file 8-30, 8-31, 8-32
allowed with PRINTER file 6-17, 6-27
allowed with SEQ file 6-29
allowed with SPECIAL file 6-31, 6-33
allowed with WORKSTN file 9-5, 9-7
file sharing 6-9
file translation 15-4
table records 15-6
first page (1P) indicator
general description 12-16
restrictions 12-16
setting 12-35
fixed currency symbol
in body of edit word 14-15
with combination edit codes 14-2
floating currency symbol
in body of edit word 14-15
with combination edit codes 14-2

flowchart
detailed program logic 5-3
fetch overflow logic 5-10, 6-23
general program logic 1-6, 1-8
lookahead logic 5-10
match fields logic 5-10, 6-34
RPG exception/error handling 5-8
RPG exception/error handling logic 4-37
format
binary 10-3
packed decimal 10-1
zoned decimal 10-2
format name 9-12
formatted dump 4-13
formatting edit words 14-14, 14-15
formatting report (auto report) 16-11
FREE (deactivate a program) operation code
program communication 11-6
full procedural file
definition GLOSS-4
function key indicators (KA-KN, KP-KY)
See also WORKSTN file
conditioning output 12-27
corresponding function keys 12-36
definition GLOSS-4
general description 12-21
setting 12-35
with WORKSTN file 9-4
function keys
corresponding indicators 12-36
with WORKSTN file 9-4

G

GDDM 11-5
general program logic 1-6, 1-8
general (01-99) indicators 12-2
generated RPG/400 program, auto report
calculations 16-16
examples 16-29
group printing 16-1
output specifications 16-17
reformatting *AUTO page headings 16-12
source of specifications 16-4
subroutine (A$$SUM) 16-16
generated specifications, auto report 16-1, 16-16
generation of program
See compilation
GENLVL parameter 3-8
GENOPT parameter A-6
glossary (definition of terms) GLOSS-1
Graphical Data Display Manager 11-5
graphics 11-5

H

half-adjust
halt (H1-H9) indicators
general description 12-22
setting 12-35
used to end a program 11-11
handling, exception/error
See exception/error handling
header specifications
See control specification
help command key 9-5
home command key 9-5
H1-H9
See halt (H1-H9) indicators

I

ICF
See Intersystem Communications Function (ICF)
ICF communications file 9-1
ID entry 4-23
If-Then-Else structure 1-10
IGC
See Double Byte Characters (DBC) variables
IGNDECERR parameter 3-14
IGNORE 6-12
example 6-13
ignoring record formats 6-12
IND entry 4-23
indexed file
access path 8-8
definition GLOSS-5
general description 8-8
indicator table, XINTAB A-4
indicators
See also individual operation codes
as exception/error indicators 4-39
control level (L1-L9)
as field record relation indicator 12-18
definition GLOSS-2
examples 12-6, 12-11
general description 12-4

indicators *(continued)*
control level (L1-L9) *(continued)*
rules for 12-5, 12-10
setting of 12-35
definition GLOSS-5
exception/error 4-39
external (U1-U8)
as field indicator 12-13
as field record relation indicator 12-18
as record identifying indicator 12-2
definition GLOSS-4
general description 12-15
resetting 12-15
rules for resetting 12-15, 12-18
setting 12-35
field
as halt indicators 12-13
general description 12-13
numeric 12-13
rules for assigning 12-13
setting of 12-35
field record relation
example 12-20
general description 12-18
rules for 12-18
file conditioning 12-17
first page (1P)
general description 12-16
restrictions 12-16
setting 12-35
function key (KA-KN, KP-KY)
See also WORKSTN file
conditioning output 12-27
corresponding function keys 12-36
general description 12-21
setting 12-35
with WORKSTN file 9-5
general description 12-3
halt (H1-H9)
as field indicator 12-13
as field record relation indicator 12-18
as record identifying indicator 12-2
as resulting indicator 12-14
general description 12-22
setting 12-35
used to end a program 11-11
last record (LR)
as record identifying indicator 12-2
as resulting indicator 12-14

indicators *(continued)*
last record (LR) *(continued)*
general description 12-16
setting 12-35
used to end a program 11-11
level zero (L0)
calculation specifications 12-22
matching record (MR)
See also multifile processing
as field record relation indicator 12-18
assigning match fields 6-34
general description 12-17
setting 12-35
output
examples 12-29, 12-30
general description 12-28
restriction in use of negative indicators 12-28
overflow
conditioning calculations 12-22
examples 6-21, 6-23
fetch overflow logic 5-11, 6-21
general description 6-17, 12-2
presence or absence of 6-19
relation to program cycle 6-23
setting of 6-23, 12-35
with PRINTER file 6-17
record identifying
assigning on input specifications 12-3
general description 12-2
rules for 12-3
setting on and off 12-35
summary 12-33
with file operations 12-2
referred to as data
*IN 12-32
*INxx 12-32
*IN,xx 12-32
return (RT)
as field indicator 12-13
as resulting indicator 12-14
conditioning output 12-27
general description 12-21
used to end a program 11-11
rules for assigning 12-3
setting of 12-35
summary chart 12-33
when set on and set off 12-35

INFDS
See file information data structure
INFSR
See file exception/error subroutine (INFSR)
input specification indicators
field indicator 12-13
field record relation indicator 12-18
record identifying indicator 12-2
input specifications 2-2
intermediate text
description A-1
listing A-1
internal indicators
first page (1P) 12-16
last record (LR) 12-16
matching record (MR) 12-17
return (RT) 12-21
interprogram communication 11-1
Intersystem Communications Function (ICF) 9-1
CFILE option 9-1
file specifications continuation line 9-1
ITDUMP parameter 3-13, A-7

K

KA-KN KP-KY indicators
See function key indicators
key
See also search argument
composite 8-7
for a record or a file 8-6
partial 8-7
key field
keyboard template xv
keyed processing
access path 8-2
examples 8-24
indexed file 8-8
random 8-21
record address limits file 8-12
sequential 8-14
sequential-within-limits 8-22
keyed sequence access path 8-2
keywords
DDS 8-1
for continuation line 8-1
See also ?
CLEAR 9-5
HELP 9-5
keywords *(continued)*
for continuation line *(continued)*
HOME 9-5
PRINT 9-5
ROLLDOWN 9-5
ROLLUP 9-5
for display device file
CLEAR 9-5
HELP 9-5
HOME 9-5
PRINT 9-5
ROLLDOWN 9-5
ROLLUP 9-5
for file information data structure (INFDS)
*FILE 4-41
*OPCODE 4-41
*RECORD 4-42
*ROUTINE 4-42
*STATUS 4-41
for program status data structure
*PARMS 4-51
*PROGRAM 4-51
*ROUTINE 4-51
*STATUS 4-50

L

language enhancements C-1
last record (LR) indicator
as resulting indicator 12-14
conditioning calculations
general description 12-16
setting 12-35
used to end a program 11-11
level checking 6-7
level zero (L0) indicator
calculation specifications 12-22
library list
definition GLOSS-5
limits records 8-3
line counter specifications 2-2
linkage to other programs
CALL/FREE/RETRN 11-1
SPECIAL device 6-29
listing, program
See program listing
literals
implied
See figurative constants

locking by RPG 6-7
record locking wait time 6-8
under commitment control 7-3
logic cycle, RPG
detail 5-3
general 1-6
logic of RPG program
detail 5-3
general 1-6
logical file
See also DISK file
definition GLOSS-5
LR (last record) indicator
See last record (LR) indicator
L0 indicator
See level zero (L0) indicator
L1-L9 (control level) indicators
See control level (L1-L9) indicators

M

match fields
See also multifile processing
alternate collating sequence 15-1
assigning values (M1-M9) to 6-35
description 6-34
dummy match field 6-36, 6-38
example 6-36, 6-37
in multifile processing 6-33
logic 5-10, 6-34
used for sequence checking 6-35
match levels (M1-M9) 6-35
matching record (MR) indicator
See also multifile processing
as field record relation indicator 12-18
general description 12-17
setting 12-35
menu, application-oriented 4-3
merging of members C-3
message retrieval 11-11
message-retrieving subroutine (SUBR23) 11-6
mixed file
See WORKSTN file
modifier statements (/COPY function)
file description specifications 16-6
input field specifications 16-7
modifying (/COPY function)
copied specifications 16-6
file description specifications 16-6
modifying (/COPY function) *(continued)*
input field specifications 16-7
monitor message (MONMSG) command 3-2
MONMSG (monitor message) command 3-2
moving double-byte data and adding control characters (SUBR41) 11-10
moving double-byte data and deleting control characters (SUBR40) 11-8
MR (matching record) indicator
See matching record (MR) indicator
multifile logic 5-10, 6-34
multifile processing
assigning match field values 6-35
logic 5-10, 6-34
match fields 6-34
no match fields 6-34
normal selection, three files 6-39, 6-40
multiple device file 9-15
multiple devices attached to application program 7-3
multiple occurrences of a data structure 10-9
M1-M9 (match field values) 6-35

N

name(s)
for *ROUTINE
with file information data structure (INFDS) 4-42
with program status data structure 4-50
negative balance (CR)
with combination edit code 14-2
with edit words 14-15
nested do group
definition GLOSS-6
nonkeyed processing 8-29
NUM entry 4-23
numeric character, definition GLOSS-6
numeric fields
auto report
centering column headings 16-13
formats 10-1, 10-3
punctuation 14-1
numeric literal
definition GLOSS-6
numeric (01-99) indicators
See field and field record relation indicators
See indicator conditioning calculations and output

numeric (01-99) indicators *(continued)*
See record identifying and resulting indicators

O

OA-OG, OV (overflow) indicators
See overflow (OA-OG, OV) indicators
open data path 6-9
operation codes 9-5, 9-7
allowed with DISK file 8-30
allowed with PRINTER file 6-17
allowed with SEQ file 6-27
allowed with SPECIAL file 6-32
OPTIMIZE 3-10
OPTION parameter 3-8, A-6
option parameter for SPECIAL PLIST 6-29
OS/400 graphics support 11-5
output
conditioning indicators 12-27
designing 1-8
indicators
See also *AUTO output specifications
examples 12-29, 12-30
general description 12-27
restrictions in use of negative indicators 12-28
output specifications 2-2
overflow indicators
conditioning calculations 12-22
conditioning output 6-19
examples 6-21, 6-23
fetch overflow logic 5-11, 6-21
general description 6-19, 12-2
presence or absence of 6-19
relation to program cycle 6-23
setting of 6-23, 12-35
with PRINTER files 6-17
override, file 6-13, 8-29
overriding external description 6-13

P

packed decimal format
array/table field 10-1
definition 10-1, GLOSS-6
description 10-1
input field 10-1
length in bytes 10-2
length in digits 10-2
packed decimal format *(continued)*
output field 10-1
page numbering
See PAGE, PAGE1-PAGE7
page overflow 6-18
parameter list
See also PARM (identify parameters) operation code
created by PARM 11-4
created by SPECIAL 6-29
definition GLOSS-6
PARM (identify parameters) operation code
program communication 11-4
rules for specifying 11-5
partial key 8-7
See also search argument
parts of an edit word
body 14-9
expansion 14-10
status 14-10
PGM parameter 3-6
PGR (Presentation Graphics Routines) 11-5
phase trace 3-13, A-7
phases
auto report program A-27
compiler A-2
PHSTRC parameter 3-13, A-7
physical file
definition GLOSS-7
PLIST keyword for SPECIAL file
description of parameters 6-29
PLIST (identify a parameter list) operation code
program communication 11-4
rules for specifying 11-4
prerunn-time array or table
See also array
definition of 13-1, 13-7, GLOSS-7
description of parameters 6-29
example of 13-6
rules for loading 13-8
Presentation Graphics Routines 11-5
print command key 9-5
printer control option
See PRTCTL
PRINTER file
access current line value 6-24
fetch overflow logic 6-21
file operation codes allowed 6-17

PRINTER file *(continued)*
maximum number of files allowed in program 6-17
modify forms control 6-24
overflow indicators 6-17
processing chart 6-27
PRTCTL (printer control) 6-24
processing
auto report program 16-26
methods
consecutive 8-15
keyed (see keyed processing)
nonkeyed 8-29
random by key 8-21
relative record number 8-14
sequential by key 8-14
sequential only 8-15, 8-29
sequential-within-limits 8-22
RPG program 16-26
WORKSTN file 9-2, 9-5
program communication
CALL/FREE/RETRN 11-1
data D-1
returning from a called program 11-11
SPECIAL 6-29
program cycle
defined 1-5
detail 5-3
general 1-6, 1-8
programmer control 5-12
program described file
as DISK file 8-8
as WORKSTN file 9-12
data format 10-1
definition GLOSS-7
program ending 5-11, 11-11
program exception/error subroutine
description 4-52
example 4-54
program exception/errors
example 4-54
general information 4-50
how to handle
default 4-50
subroutine (*PSSR) 4-52
indicators in positions 56 and 57 of calculation specifications 4-50
data structure 4-50
status information 4-50
program identification
See program name
program listing
example 4-3
program logic
detail 5-3
general 1-6, 1-8
program name
*PGM parameter 3-7
program status data structure
general information 4-50, 10-9
multiple occurrences of 10-9
predefined subfield 4-33
predefined subfields 4-50
statement
externally described 10-6
program described 10-6
rules 10-8
specifications 10-8
status codes 4-51
subfields
predefined 4-33, 4-50
rules 10-12
specifications 10-11
program template A-1
programmer control of file processing 5-12
programming operation codes, structured 1-18
protecting records/files
See file locking by RPG
PRTCTL (printer control)
example 6-26
general information 6-24
PRTFILE parameter 3-11

Q

QRGAE compiler phase A-3
QRGCC compiler phase A-3
QRGCR compiler phase A-3
QRGC1 compiler phase A-3
QRGDI compiler phase A-3
QRGEC compiler phase A-3
QRGGB compiler phase A-3
QRGGC compiler phase A-3
QRGGI compiler phase A-3
QRGGO compiler phase A-3
QRGGS compiler phase A-3

QRGIC compiler phase A-3
QRGMC compiler phase A-3
QRGRT compiler phase A-3
QRGSC compiler phase A-3
QRGSE compiler phase A-3
QRGSF compiler phase A-3
QRGSI compiler phase A-3
QRGSO compiler phase A-3
QRGYC compiler phase A-3
QRG1 compiler phase A-3
QRPT0000 A-27
QRPT0001 A-27
QRPT0002 A-27
QRPT0003 A-27
QRPT0004 A-27
QRPT0005 A-27
QRPT0008 A-27

R

random by key processing
  description 8-21
  example 8-23
random by relative record number processing 8-14
RCLRSC 6-9
READ (read a record) operation code
  with data communication D-1
READC (read next modified record) operation code
  with WORKSTN subfile 9-9
RECNO
  with relative record number processing 8-14
record
  limits 8-13
  releasing 6-8
  renaming 6-10
record address file
  relative record number 8-12
  sequential-within-limits 8-12
  with limits records 8-13
  with relative record numbers 8-12
record address limits file
  See record address file
record address relative record number file
  See record address file
record format
  for a subfile 9-8
  specifications for externally described files 8-1
record identifying indicators (01-99, H1-H9, L1-L9, LR, U1-U8, RT)
  assigning on input specifications
    for externally described file 12-4
    for program described file 12-4
    rules for 12-3
  general description 12-2
  setting on and off 12-35
  summary 12-33
  with file operations 12-2
record name table, XRCTAB A-4
record sharing
  See file locking by RPG
records, alternate collating sequence table 15-4
records, file translation table 15-5
redirection, file
  definition 6-2
  general description 6-2
relative record number processing 8-14
relative record number record address file
  See record address file
releasing records 6-8
RENAME 6-10
  example 6-12
renaming record format names 6-10
requester
  in INFDS 4-25, 4-30
reserved words
  definition GLOSS-8
  *CANCL 4-46, 5-3
  *DETC
    file exception/error subroutine (INFSR) 4-46
    file information data structure (INFSR) 4-42
    flowchart 5-3
    program exception/errors 4-50
  *DETL
    file exception/error subroutine (INFSR) 4-46
    file information data structure (INFDS) 4-42
    flowchart 5-2
    program exception/errors 4-51
  *FILE 4-41
  *GETIN
    file exception/error subroutine (INFSR) 4-46
    file information data structure (INFDS) 4-37, 4-42
    flowchart 5-2, 5-8

reserved words *(continued)*
*GETIN *(continued)*
program exception/errors 4-51
*IN 12-32
*INIT 4-42, 4-51
*INxx 12-32
*IN,xx 12-32
*OFL
file exception/error subroutine (INFSR) 4-46
file information data structure (INFDS) 4-42
flowchart 5-3
program exception/errors 4-51
*OPCODE 4-41
*PARMS 4-51
*PROGRAM 4-51
*RECORD 4-42
*ROUTINE 4-42, 4-51
*STATUS 4-41, 4-50
*TERM 4-42, 4-51
*TOTC
file exception/error subroutine (INFSR) 4-46
file information data structure (INFDS) 4-42
flowchart 5-3
program exception/errors 4-51
*TOTL
file exception/error subroutine (INFSR) 4-46
file information data structure (INFDS) 4-42
flowchart 5-3
program exception/errors 4-51
resulting indicators (01-99, H1-H9, OA-OG, OV, L1-L9, LR, U1-U8, KQ-KN, KP-KY, RT)
See also individual operation codes
as exception/error indicators 4-39
general description 12-14
rules for assigning 12-14
setting of 12-35
retrieval of data area
implicit 5-2, 10-9
RETRN (return to caller) operation code
program communication 11-11
return points
for file exception/error subroutine
blanks 4-46
*CANCL 4-46
*DETC 4-46
*DETL 4-46
return points *(continued)*
for file exception/error subroutine *(continued)*
*GETIN 4-46
*OFL 4-46
*TOTC 4-46
*TOTL 4-46
for program exception/error subroutine 4-52
return status parameter 6-30
return (RT) indicator
as resulting indicator 12-14
general description 12-21
setting of 12-35
used to end a program 11-11
RETURNCODE data area 3-2
returning from a called program
methods of 11-11
rolldown command key 9-5
rollup command key 9-5
RPG
auto report program phases A-27
compiler phases A-2
default error handler 4-35
processing 2-1
syntax checker 2-3
RPG enhancements
language C-1
RPG logic cycle
detail 5-3
general 1-6, 1-8
RPG restrictions, summary E-1
RPGOBJ 3-7
RT (return) indicator
See return (RT) indicator
run-time array
See also array
definition of 13-2, GLOSS-8
rules for loading 13-3
with consecutive elements 13-4
with scattered elements 13-5
run-time subroutines A-4

S

sample program B-1
SAVDS entry 4-23
search argument
externally described file
description 8-6
referencing a partial key 8-7

search argument *(continued)*
externally described file *(continued)*
valid 8-6
program described files 8-8
SEQ file
definition GLOSS-8
example 6-28
file operation codes allowed 6-28
general description 6-27
processing chart 6-29
restrictions 6-27
variable-length 6-27
sequence
collating
See alternate collating sequence
sequence checking
alternate collating sequence 15-1
on input specifications 6-17
sequential by key processing
description 8-14
example 8-17
sequential file 8-9
sequential within limits processing
description 8-21
examples 8-22, 8-24
sequential-only processing 8-15
SEU (source entry utility) 2-3
SFILE
See also subfiles
descriptions 9-7
sharing an open data path for a file 6-9
signs
external format 10-5
internal format 10-5
simple edit codes (X, Y, Z) 14-1
SNPDUMP parameter 3-13, A-7
source and cross-reference listing example 4-3
source entry utility (SEU) 2-3
source program
auto report 16-26
general information 16-27
RPG
compiling 3-1
entering into system 2-3
listing A-6
running 4-3
special command keys 9-5
SPECIAL file
definition 6-29
device name 6-29
file operation codes allowed with 6-31
general description 6-29
parameter list 6-29
specification forms
split control field 12-11
spooling
output 6-3
SQL
See Structured Query Language (SQL)
SRCFILE parameter 3-7
SRCMBR parameter 3-7
status codes
in file information data structure (INFDS) 4-42
in program status data structure 4-51
status parameter for SPECIAL PLIST 6-29
status (of an edit word) 14-13
STRSEU (edit source) command 2-3
structured programming operation codes
Structured Query Language (SQL) 1-4
definition GLOSS-8
entering statements into RPG 1-4
subfields
for data structure subfield specifications 10-11
for file information data structure 4-13, 4-39
for program status data structure 4-33, 4-50
for PRTCTL 6-24
subfields within a data structure
description 10-11
rules for 10-12
specifications for 10-11
subfiles
control record format 9-7
definition GLOSS-8
examples 9-10, 9-11
file operation codes allowed with 9-9
general description 9-7, 9-9
record format 9-7
uses of 9-10
subroutines
A$$SUM (auto report) 16-16
file exception/error (INFSR) 4-45
program exception/error (*PSSR) 4-52
run-time A-4
SUBR23R3 (message retrieval) 11-11

subroutines *(continued)*
SUBR40R3 (manipulating DBC variables) 11-11
SUBR41R3 (manipulating DBC variables) 11-11
SUBR23 (message retrieving subroutine) 11-6
SUBR23R3 (message retrieval) 11-11
SUBR40 (moving double-byte data and deleting control characters) 11-8
SUBR40R3 (manipulating Double Byte Characters variables) 11-11
SUBR41 (moving double-byte data and adding control characters) 11-10
SUBR41R3 (manipulating Double Byte Characters variables) 11-11
summary tables
DISK file processing 8-32, 8-34
edit codes 14-4, 14-5
file operation codes allowed with
DISK 8-29
PRINTER 6-17, 6-27
SEQ 6-28
SPECIAL 6-31, 6-33
WORKSTN 9-5, 9-7
function key indicators and corresponding function keys 12-36
indicators 12-33, 12-35
PRINTER file processing 6-27
RPG restrictions E-1
SEQ file processing 6-29
SPECIAL file processing 6-33
WORKSTN file processing 9-7
suppression of zero
in body of edit word 14-11
with combination edit codes 14-2
syntax checker, RPG 2-3

T

T-*AUTO
See *AUTO output specifications
table
See also array
defining 13-26
definition of terms 13-1
differences from array 13-1
element, specifying 13-27
example of using 13-28
loading 13-26
table *(continued)*
LOKUP operation
with one table 13-27
with two tables 13-27
searching
See LOKUP operation
specifying a table element 13-27
tape file 8-9
See also SEQ file
techniques for efficient coding 1-9
template, program A-1
testing fields
See field indicators
testing RPG programs
breakpoint 4-6
test library 4-4
trace 4-10
TEXT parameter 3-8
trace 4-10
translation table and alternate collating sequence coding sheet 15-1
translation, file
See file translation
triple asterisk (***)
generated specifications 16-1, 16-16

U

Unblocking input records 6-8
unwanted control breaks 12-5, 12-7, 12-9
updating records/files
See file blocking by RPG
usage of indicators
See indicators
user-defined edit codes (5-9) 14-4
using auto report 16-1
using, displaying, and printing messages 4-1
USRPRF parameter 3-12
U1-U8
See external (U1-U8) indicators

V

variable-length records 6-27
variables
ideographic 11-11
VCOMMON A-4

W

WORKSTN file
  definition 9-1
  examples 9-16
  file operation codes allowed with 9-5
  function key indicators with 9-4
  processing
    description 9-5
  program described
    example 9-56
    with format name 9-12
    without format name 9-14
  subfiles
    control record format 9-7
    definition GLOSS-8
    examples 9-10
    for display device file 9-7
    record format 9-7
    uses of 9-10
WRITE (create new records) operation code
  with data communication D-1
  with WORKSTN subfile 9-23

X

XFDTAB A-4
XFLTAB A-4
XINTAB A-4
XRCTAB A-4

Y

Y edit code

Z

zero suppression 14-2
  in body of edit word 14-12
  with combination edit code 14-2
zoned decimal format
  definition 10-3, GLOSS-9
  description 10-3

Numerics

01-99 indicators
  See field and field record relation indicators
  See indicators conditioning calculations and output
  See record identifying indicators and resulting indicators
1P (first page) indicator
  general description 12-16
  restrictions 12-16
  setting 12-35

Special Characters

& (ampersand)
  auto report copy function 16-7
  use in edit word 14-10, 14-13
$ (fixed or floating currency symbol)
  in body of edit word 14-12
  with combination edit codes 14-1
* (asterisk)
  generated specifications 16-1, 16-16
  in body of edit word 14-11
  with combination edit codes 14-1
*AUTO output specifications
  See also auto report function
  field description specifications
  generated end positions 16-11
  generated RPG/400 specifications 16-16
  group printing 16-1
  report format 16-11
  spacing and skipping 16-11
*AUTO specifications
  See *AUTO output specifications
*CANCL 4-46, 5-3
*DETC
  file exception/error subroutine (INFSR) 4-46
  file information data structure (INFDS) 4-42
  flowchart 5-3
  program exception/errors 4-51
*DETL
  file exception/error subroutine (INFSR) 4-46
  file information data structure (INFDS) 4-42
  flowchart 5-2
  program exception/errors 4-51
*DUMP
  GENOPT parameter value 3-9, A-6
  OPTION parameter value 3-9, A-6
*ENTRY PLIST 11-4
*EQUATE 15-6
*FILE 4-41
*FILEbb 15-5
*GETIN
  file exception/error subroutine (INFSR) 4-46
  file information data structure (INFDS) 4-42
  flowchart 5-2

*GETIN *(continued)*
program exception/errors 4-51
*IN 12-32
*INIT 4-42, 4-51
*INP 4-41
*INxx 12-32
*IN,xx 12-32
*LIST A-6
*MODE 4-41
*NOOPTIMIZE 3-10, A-6
*OFL
file exception/error subroutine (INFSR) 4-46
file information data structure (INFDS) 4-42
flowchart 5-3
program exception/errors 4-51
*OPCODE 4-41
*OPTIMIZE A-6
*OUT 4-41
*PARMS 4-51
*PROGRAM 4-51
*PSSR
See program exception/error subroutine
*RECORD 4-42
*ROUTINE
with file information data structure (INFDS) 4-42
with program status data structure 4-51
*SIZE 4-41
*SOURCE A-6
*STATUS
with file information data structure (INFDS) 4-41
with program status data structure 4-50
*TERM 4-42, 4-51
*TOTC
file exception/error subroutine (INFSR) 4-42
file information data structure (INFDS) 4-42
flowchart 5-3
program exception/errors 4-46
*TOTL
file exception/error subroutine (INFSR) 4-46
file information data structure (INFDS) 4-42
flowchart 5-3
program exception/errors 4-51
*XREF
GENOPT parameter value 3-9, A-6
OPTION parameter value 3-9, A-6
**
alternate collating sequence table 15-4
arrays and tables 13-6
file translation table 15-4
generated specifications 16-1, 16-16
***
generated specifications 16-1, 16-16
/COPY statement specifications
See also auto report function
format of 16-5
modifying copied specifications 16-6
file description 16-6
input field 16-7
order of specifications included 16-5
placement in auto report source program 16-5
sorting of specifications 16-5
/END EXEC delimiter 1-4
/EXEC SQL delimiter 1-4

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your nearest IBM branch office. You may use this form to communicate your comments about this publication, its organization, or subject matter **with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.**

☐ If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your nearest IBM branch office to request additional publications.

Name _____

Company or Organization _____

Address _____

_____

SC09-1161-00

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Development Laboratory
Information Development, Department 245
Rochester, Minnesota 55901

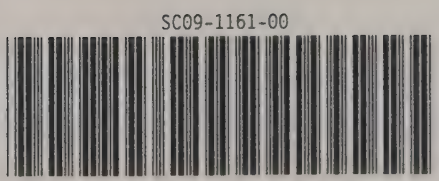
Fold and tape | Please Do Not Staple | Fold and tape

IBM®

International Business Machines Corporation

IBM

Program Number
5728-RG1



Printed in U.S.A.

IBM

AS/400™

SC21-9609-0

Programming: Structured Query Language/400 Programmer's Guide

IBM

AS/400™

SC21-9609-0

Programming:
Structured Query Language/400
Programmer's Guide

First Edition (October 1988)

This edition applies to Release 1 Modification Level 2 of IBM Structured Query Language/400 (SQL/400) Licensed Program (Program 5728-ST1), and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

Changes are periodically made to the information herein; any such changes will be reported in subsequent revisions or technical newsletters.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

The numbers at the bottom right of illustrations are publishing control numbers and are not part of the technical content of this manual.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to your IBM-approved remarketer.

This publication could contain technical inaccuracies or typographical errors.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department 245, Rochester, Minnesota, U.S.A. 55901. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

AS/400, COBOL/400, Operating System/400, OS/400, RPG/400, and SQL/400 are trademarks of the International Business Machines Corporation.

© Copyright International Business Machines Corporation 1988

About This Manual

This manual shows programmers and data base managers how to use the IBM AS/400 Structured Query Language/400 (SQL/400), how to access data in a data base, and how to prepare, run, and test an application program containing SQL statements.

Who Should Use This Manual

This manual is intended for application programmers and data base managers who are familiar with COBOL/400, AS/400 PL/I, and/or RPG III (part of RPG/400) languages and who understand basic data base applications.

What You Should Know

You should know how to use and write application programs for the AS/400 system. You also should know how to program with COBOL/400, AS/400 PL/I, or RPG III.

Rules Relating to Examples of SQL Statements

The examples of SQL statements shown in this manual are based on the sample tables in Appendix B and assume the following:

- They are shown in the interactive SQL environment or written in COBOL. EXEC SQL and END-EXEC are used to delimit an SQL statement in a COBOL program. A description of how to use SQL statements in a COBOL program is provided in Chapter 5.
- Each SQL example is shown on several lines, with each clause of the statement on a separate line.
- SQL reserved words are highlighted.
- SQL naming convention is used.
- The APOST and APOSTSQL precompiler options are assumed (although they are not the default in COBOL). Character string literals within SQL and host language statements are delimited by apostrophes (').
- The complete syntax of the SQL statement is usually not shown in any one example. For the complete description and syntax of any of the statements described in this manual, refer to the *SQL/400 Reference* manual.

Whenever the examples vary from these assumptions, it is stated.

Because this manual is intended for the application programmer, most of the examples are shown as if they were written in an application program. However, many examples can be slightly changed and run interactively, using the interactive SQL (see Chapter 10). The syntax of an SQL statement, when using interactive SQL, differs slightly from the format of the same statement when it is embedded in a program.

How This Manual Is Organized

Chapter 1 introduces you to the Structured Query Language (SQL), explaining the concepts and objects of SQL. A comparison of AS/400 system and SQL naming rules is shown, and the SQL catalog is described.

Chapter 2 describes how to create and work with data bases, tables, and views, and how to use catalogs in data base design.

Chapter 3 describes coding techniques for using basic SQL statements and clauses, using a cursor, performing complex search conditions, and joining data from more than one table.

Chapter 4 provides common concepts and rules for using SQL with host languages.

Chapter 5 describes using SQL statements in COBOL programs.

Chapter 6 describes using SQL statements in RPG III programs.

Chapter 7 describes using SQL statements in PL/I programs.

Chapter 8 describes how to issue SQL statements that are defined and run at program run time.

Chapter 9 describes how to prepare and run a program containing SQL statements.

Chapter 10 describes how to use interactive SQL to syntax check, validate, and run SQL statements.

Chapter 11 describes the security plan for protecting SQL data.

Chapter 12 describes how to establish a test environment for and how to debug SQL statements in an application program.

Chapter 13 provides guidelines and techniques for fine-tuning SQL statements in an application program and for working with retrieved data.

Appendix A contains a list of reserved words for SQL/400.

Appendix B contains sample tables and the statements for creating them for inserting information into them. The examples used throughout this manual are based on these sample tables.

Appendix C contains a description of SQL return codes.

Appendix D contains sample programs using SQL/400 statements.

This manual also contains a glossary of terms and abbreviations and an index. Use the glossary to find the meaning of unfamiliar terms. Use the index to look up a topic.

Related Online Information

The following online information is available on the AS/400 system. You can press the Help key a second time to see an explanation of how the online information works, including the index search function.

Help for Displays

You can press the Help key on any display to see information about the display. There are two types of help available:

General
Specific

General help explains the purpose of the display. General help appears if you press the Help key when the cursor is outside the areas for which specific help is available.

Specific help explains the field on which the cursor is positioned when you press the Help key. For example, it describes the choices available for a prompt. If a system message appears at the bottom of the display, position the cursor on the message and press the Help key to see information about the cause of the message and the appropriate action to take.

To exit the online information, press F3 (Exit). You return to the display on which you pressed the Help key.

Index Search

Index search allows you to specify the words or phrases you want to see information about. To use index search, press the Help key, then press F11 (Search index).

Help for Control Language Commands

To see prompts for parameters for a control language (CL) command, type the command name, then press the Help key or F4 (Prompt) instead of the Enter key.

Online Education

AS/400 system online education provides tutorials on a wide variety of topics. To use the online education, press F13 (User support) on any system menu to show the User Support menu. Then select the option to use online education.

Question-and-Answer Function

The question-and-answer (Q & A) function provides answers to questions you may have about using an AS/400 system. To use the Q & A function, press F13 (User support) on any system menu to show the User Support menu. Then select the option to use the question-and-answer function.

Related Printed Information

If you need more information about using SQL statements, statement syntax and parameters, the see the following manuals:

- *Programming: Structured Query Language/400 Reference*, SC21-9608
- *Systems Application Architecture Common Programming Interface Database Reference*, SC26-4348.

If you need more information about the interactive data definition utility, see the *Utilities: Interactive Data Definition Utility User's Guide*, SC21-9657.

For more information about AS/400 system security, see the *Programming: Security Concepts and Planning*, SC21-8083.

For more information about entering source and syntax checking of host language and SQL statements, see the *Application Development Tools: Source Entry Utility User's Guide and Reference*, SC09-1172.

For more information on AS/400 control language commands and system programming, see the following:

- *Languages: COBOL/400 User's Guide*, SC09-1158
- *Languages: PL/I Reference Summary*, SX09-1051
- *Languages: PL/I User's Guide and Reference*, SC09-1156
- *Languages: RPG/400 User's Guide*, SC09-1161
- *Programming: Command Reference Summary*, SC21-8076
- *Programming: Control Language Programmer's Guide*, SC21-8077
- *Programming: Control Language Reference*, SBOF-0481

For more information about data bases, see the following:

- *Programming: Backup and Recovery Guide*, SC21-8079
- *Programming: Data Base Guide*, SC21-9659
- *Programming: Data Description Specifications Reference*, SC21-9620

Contents

Chapter 1. Introduction to SQL 1-1
SQL Concepts 1-1
Relational Data Bases and Terminology 1-2
Types of SQL Statements 1-2
SQL Objects 1-3
Data Bases 1-3
Data Dictionary 1-3
Journals and Journal Receivers 1-3
Tables, Rows, and Columns 1-3
Views 1-4
Indexes 1-5
Catalogs 1-5
Catalog Views 1-6
SYSCOLUMNS 1-6
SYSINDEXES 1-8
SYSKEYS 1-8
SYSTABLES 1-9
SYSVIEWDEP 1-9
SYSVIEWS 1-10
Compiled Application Program Objects 1-10
User Source File Member 1-10
Temporary Source File Member 1-11
Program 1-11

Chapter 2. Working with SQL Data Bases, Tables, Views, Indexes, and Catalogs 2-1
Creating an SQL Data Base 2-1
Creating and Using a Table 2-1
Creating the Department Table (TDEPT) 2-2
Using the LABEL ON Statement for Defining Table Names and Column Headings 2-2
Inserting Information into a Table 2-2
Getting Information from a Single Table 2-4
Getting Information from More Than One Table 2-5
Changing Information in a Table 2-6
Deleting Information in a Table 2-7
Creating and Using a View 2-8
Creating a View on a Single Table 2-8
Creating a View Combining Data from More Than One Table 2-9
Notes on Using a View 2-10
Working with Indexes 2-11
Using the Catalog in Data Base Design 2-11
Getting Catalog Information about a Table 2-11
Getting Catalog Information about a Column 2-11
Getting Catalog Information about Indexes 2-12
Getting Catalog Information about Views 2-12
Building a View over Catalog Views 2-12
Using COMMENT ON 2-13
Getting Comments 2-13

Chapter 3. SQL Coding Techniques 3-1
Using Basic SQL Statements and Clauses 3-1
The INSERT Statement 3-1

The UPDATE Statement . . . 3-3
The DELETE Statement . . . 3-4
The SELECT INTO Statement . . . 3-5
The WHERE Clause . . . 3-8
The GROUP BY Clause . . . 3-11
The HAVING Clause . . . 3-12
The ORDER BY Clause . . . 3-14
Using the UNION Keyword to Combine SELECT Statements . . . 3-16
Restrictions for the Length and Data Type of Columns . . . 3-18
Specifying UNION ALL . . . 3-19
Using a Cursor . . . 3-20
Example of How to Use a Cursor . . . 3-20
A Unit of Recovery and Open Cursors . . . 3-25
Preventing Duplicate Rows . . . 3-25
Performing Complex Search Conditions . . . 3-26
Keywords for Use in Search Conditions . . . 3-26
Joining Data from More Than One Table . . . 3-29
Join and the WHERE Clause . . . 3-31
Notes on the Join Technique . . . 3-33
Inserting Multiple Rows into a Table . . . 3-34
Inserting Default Values into Columns . . . 3-35

Chapter 4. Common Concepts and Rules for Using SQL with Host Languages 4-1
Using Host Variables in SQL Statements . . . 4-1
Assignment Rules . . . 4-2
Indicator Variables and Structures . . . 4-4
Handling SQL Error Return Codes . . . 4-6
Handling Exception Conditions with the WHENEVER Statement . . . 4-7

Chapter 5. Using SQL Statements in COBOL Programs . . . 5-1
Application Requirements . . . 5-1
SQL Communication Area (SQLCA) . . . 5-1
Coding Requirements . . . 5-2
Host Structures . . . 5-3
Basic Requirements for Host Variables . . . 5-4
Coding Rules for Host Variables . . . 5-5
Assignment Rules . . . 5-5
Allowable COBOL Declarations for Host Variables . . . 5-5
Indicator Variables and Structures . . . 5-8
Handling SQL Error Return Codes . . . 5-8

Chapter 6. Using SQL Statements in RPG Programs . . . 6-1
Application Requirements . . . 6-1
SQL Communication Area (SQLCA) . . . 6-1
Coding Requirements . . . 6-2
Host Structures . . . 6-4
Basic Requirements for Host Variables . . . 6-5
Coding Rules for Host Variables . . . 6-5
Assignment Rules . . . 6-6
Indicator Variables and Structures . . . 6-6
External Descriptions . . . 6-6
The WHENEVER Statement . . . 6-7
Handling SQL Error Return Codes . . . 6-7

Chapter 7. Using SQL Statements in PL/I Programs . . . 7-1
Application Requirements . . . 7-1

SQL Communication Area (SQLCA) . . . 7-1
SQL Descriptor Area (SQLDA) . . . 7-1
Coding Requirements . . . 7-2
Using PL/I Host Variables in SQL Statements . . . 7-3
Host Structures . . . 7-3
Basic Requirements for Host Variables . . . 7-4
Coding Rules for Host Variables . . . 7-4
Assignment Rules . . . 7-5
Allowable PL/I Declarations for Host Variables . . . 7-5
Indicator Variables and Structures . . . 7-7
Using the %INCLUDE Directive for External File Descriptions . . . 7-8
Element Description . . . 7-8
Structure Definition . . . 7-9
Structure Ending . . . 7-10
%INCLUDE Example . . . 7-11
Handling SQL Error Return Codes . . . 7-11

Chapter 8. Dynamic SQL Applications . . . 8-1
Designing and Running a Dynamic SQL Application . . . 8-3
Processing NonSELECT Statements . . . 8-3
Using the PREPARE and EXECUTE Statements . . . 8-4
Processing SELECT Statements and Using SQLDA . . . 8-5
Fixed-List SELECT Statements . . . 8-5
Varying-List SELECT Statements . . . 8-6
The SQL Descriptor Area (SQLDA) . . . 8-7
SQLDA Format . . . 8-7
Example of a SELECT Statement for Allocating Storage for SQLDA . . . 8-10
Using a Cursor . . . 8-14
Using Parameter Markers . . . 8-15

Chapter 9. Preparing and Running a Program with SQL Statements . . . 9-1
Basic Processes of the SQL Precompiler . . . 9-1
Input to the Precompiler . . . 9-2
Output from the Precompiler . . . 9-2
Precompiler Commands . . . 9-8
Syntax for the Precompiler Commands . . . 9-8
Precompiler Command Parameters . . . 9-12
Parameter Definitions . . . 9-12
Required Parameter . . . 9-12
Optional Parameters . . . 9-13
Example of the Precompiler Source Command . . . 9-16
Compiling an Application Program . . . 9-16
Binding an Application . . . 9-17
Program References . . . 9-18
Running a Program with Embedded SQL . . . 9-18
Blocking Considerations . . . 9-18
OS/400 DDM Considerations . . . 9-19
Override Considerations . . . 9-19
SQL Return Codes . . . 9-19

Chapter 10. Using Interactive SQL . . . 10-1
Overview . . . 10-1
Terminology . . . 10-2
Getting Started . . . 10-3
Functional Description . . . 10-3
Statement Entry . . . 10-3

Session Services 10-4
List Selection Function 10-5
Exit Interactive SQL 10-6
Help 10-6
The Session and Its Functions 10-7
Recovering a Failed SQL Session 10-7
Messages 10-7
Supported SQL Statements 10-8
Interactive Session Display Flow Diagram 10-8
Tips on Using Interactive SQL 10-11
Using the List Selection Function 10-11
Testing Your SQL Statements Using Interactive SQL 10-14
Entering DBCS Data 10-14
STRSQL Command 10-15
Example 10-16

Chapter 11. SQL Data Protection 11-1
SQL Security 11-1
Authorization ID 11-1
Public Authority 11-1
Views 11-1
SQL Data Integrity 11-2
Concurrency 11-2
Atomic Operations 11-3
Journaling 11-3
Commitment Control 11-4
Save/Restore 11-5
Damage Tolerance 11-5
Index Recovery 11-5

Chapter 12. Testing SQL Statements in Application Programs 12-1
Establishing a Test Environment 12-1
Designing a Test Data Structure 12-1
Debugging Your Program 12-2

Chapter 13. Guidelines and Techniques for Using SQL 13-1
Guidelines for Using SQL Statements 13-1
Effectively Using an SQL Index 13-2
Improving Performance When Selecting Data from Two or More Tables . 13-5
Improving Performance by Reducing the Number of Opens 13-6
Improving Performance when Paging Interactively Displayed Data 13-6
Techniques for Solving Some Common Data Base Problems 13-7
Paging through Retrieved Data 13-7
Keeping a Copy of the Data 13-7
Retrieving Data a Second Time 13-7
Establishing Position at the End of a Table 13-9
Adding Data to the End of a Table 13-9
Updating Data as It Is Retrieved from the Data Base 13-9
Updating Data Previously Retrieved 13-10
Changing the Table Definition 13-10

Appendix A. SQL Reserved Words A-1

Appendix B. SQL Sample Tables B-1
Creating the Tables B-1
Creating the Department Table (TDEPT) B-1

Creating the Employee Table (TEMPL) B-2
Creating the Project Table (TPROJ) B-2
Creating the Employee Project Account Table (TEMPRACT) B-2
Inserting Information into the Tables B-3
Inserting Information into the TDEPT Table B-3
Inserting Information into the TEMPL Table B-3
Inserting Information into the TPROJ Table B-4
Inserting Information into the TEMPRACT Table B-4
Sample Tables B-5

Appendix C. SQLCODES C-1
SQLCODE Descriptions C-2
Positive SQLCODEs C-2
Negative SQLCODEs C-2

Appendix D. Sample Programs Using SQL/400 Statements D-1
SQL Statements in COBOL Programs D-3
SQL Statements in PL/I Programs D-11
SQL Statements in RPG Programs D-17
Report Produced by Sample Programs D-23

Glossary G-1

Index X-1

Chapter 1. Introduction to SQL

This manual describes the AS/400[1] system implementation of the Structured Query Language (SQL). SQL manages information based on the relational model of data. SQL statements may be embedded in high-level languages or may be run interactively.

SQL consists of statements and clauses that describe what you want to do with the data in a data base and under what conditions you want to do it.

SQL Concepts

SQL/400[1] consists of three main parts:

- SQL parser and run-time support

  This part supplies the parsing of SQL statements and the support to run any SQL statement. SQL/400 interfaces with the existing system functions to use SQL statements. This support is part of the Operating System/400 (OS/400)[1], which allows applications that contain SQL statements to be run on systems where SQL is not installed.

- SQL precompilers

  This part supports precompiling embedded SQL statements in host languages. Three languages are supported: COBOL/400[1], AS/400 PL/I, and RPG III (part of RPG/400)[1]. The SQL host language precompilers prepare an application program containing SQL statements. The host language compilers then compile the precompiled host source programs. For more information on precompiling, see Chapter 9.

- SQL interactive interface

  This part supplies you with an interactive interface for creating and running SQL statements. For more information on interactive SQL, see Chapter 10.

---

1 AS/400, COBOL/400, Operating System/400, OS/400, RPG/400, and SQL/400 are trademarks of the International Business Machines Corporation.

Relational Data Bases and Terminology

In the relational model of data, all data is perceived as existing in tables. SQL/400 objects are created and maintained as AS/400 system objects. The following table shows the relationship between AS/400 system terms and SQL relational data base terms. For more information on data bases, see the *Data Base Guide*.

System Terms	SQL Terms
Library. Groups related objects and allows the user to find the objects by name.	**Data base**. Consists of a library, a journal, a journal receiver, a data dictionary, and an SQL catalog. A data base groups related objects and allows the user to find the objects by name.
Physical file. A collection of records.	**Table**. A collection of columns and rows.
Record. A collection of fields.	**Row**. The horizontal part of a table containing a serial collection of columns.
Field. One or more characters of related information of one data type.	**Column**. The vertical part of a table of one data type.
Logical file. A subset of fields and records of one or more physical files.	**View**. A subset of columns and rows of one or more tables.

Types of SQL Statements

There are two basic types of SQL statements: data definition statements (DDL) and data manipulation statements (DML). SQL data definition statements can only operate on objects created by SQL in an SQL data base. SQL data manipulation statements can operate on objects created by SQL as well as AS/400 physical files and AS/400 single-format logical files, whether or not they reside in an SQL data base.

The following SQL statements are data definition statements:

COMMENT ON
CREATE DATABASE
CREATE INDEX
CREATE TABLE
CREATE VIEW
DROP
GRANT
LABEL ON
REVOKE

The following SQL statements are data manipulation statements:

CLOSE
COMMIT
DECLARE CURSOR
DELETE
FETCH
INSERT
LOCK TABLE
OPEN
ROLLBACK
SELECT
UPDATE

SQL Objects

SQL objects used on the AS/400 system are data bases, tables, views, indexes, and catalogs. SQL/400 creates and maintains these objects as AS/400 data base objects. A brief description of these objects follows.

Data Bases

A **data base** consists of a library, a journal, a journal receiver, an catalog, and a data dictionary. Tables, views, and other system objects (such as programs) can be created, moved, or restored in an SQL data base. SQL tables, views, and indexes can only be created into an AS/400 data base. They cannot be created into an AS/400 library.

AS/400 physical files can be placed in a created, moved, or restored SQL data base. AS/400 logical files may not be placed in an SQL data base because they cannot be described in the data dictionary.

You may create and own many data bases.

Data Dictionary

A **data dictionary** is a set of tables containing object definitions.

SQL automatically creates a data dictionary when a data base is created. The dictionary is then automatically maintained by the system. You can work with data dictionaries using the interactive data definition utility (IDDU), which is part of the OS/400. For more information on IDDU, see the *IDDU User's Guide*.

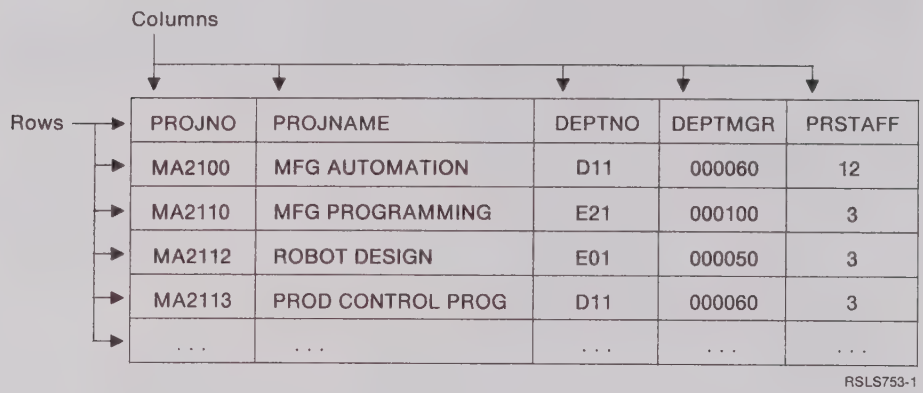
Journals and Journal Receivers

A **journal** and **journal receiver** are used to record changes to tables and views in the data base. The journal and journal receiver are then used in processing of SQL COMMIT and ROLLBACK statements. For more information on journaling, see the *Backup and Recovery Guide*.

Tables, Rows, and Columns

A **table** is a two-dimensional arrangement of data consisting of **rows** and **columns**. The row is the horizontal part containing one or more columns. The column is the vertical part containing one or more rows of data of one data type. All data for a column must be of the same type. A table in SQL/400 is a nonkeyed physical file. See the section "Data Types" in the *SQL/400 Reference* manual for a description of data types.

The following is a sample SQL table:



SQL Data Types

When you create a table in SQL, you define each of its columns to hold one of the following types of data:

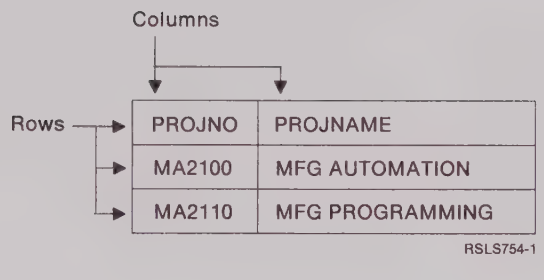
CHARACTER	Any EBCDIC character.
DECIMAL	A packed decimal number with an implied decimal point.
REAL	A short (4-byte) floating-point number in IEEE format.
FLOAT	An 8-byte floating point number in IEEE format.
INTEGER	A binary number with a precision of 31 bits.
NUMERIC	A zoned decimal number with an implied decimal point.
SMALLINT	A binary number with a precision of 15 bits.

For more information on data types, see the section on "Data Types" in the *SQL/400 Reference* manual.

Views

A **view** appears like a table to an application program; however, a view contains no data. It is created over one or more tables stored on the data base. A view can contain all the columns of a given table(s) or some subset of them, or can contain all the rows of a given table(s) or some subset of them. The columns may be arranged differently in a view than they are in the table(s) from which they are taken. A view in SQL/400 is a special form of a nonkeyed logical file.

The following figure shows a view created from the preceding example of a SQL table. Notice that the view is created only over the *PROJNO* and *PROJNAME* columns of the table and for rows MA2110 and MA2111.



Indexes

An SQL **index** is a collection of the data in the columns of a table that are logically arranged in either ascending or descending order. Each index contains a separate arrangement. An SQL/400 index is a keyed logical file.

The index is used by the system for faster data retrieval. Whether you create an index is optional. You can create any number of indexes. You can create or drop an index at any time. The index is automatically maintained by the system. However, because the indexes are maintained by the system, a large number of indexes can adversely affect the performance of applications that change the table.

Catalogs

An SQL **catalog** consists of a set of views and logical files based on:

- Two data base files in QSYS (maintained by the AS/400 data base manager) containing cross-reference information on:
  - The relationships between files and dictionaries
  - The relationships between files
- The set of data dictionary files in the data base containing object definitions

Catalog views only contain information about objects in one data base. The information in a catalog is about your SQL data base and its contents.

The catalog describes every table, view, index, and file in the data base and includes column definitions. However, there are four logical files existing in every data base that are not described in the catalog. The information can be queried like tables.

A catalog is automatically created when you create a data base. You cannot drop or explicitly change the catalog.

The views contained in the SQL catalog are named:

SYSCOLUMNS
SYSINDEXES
SYSKEYS
SYSTABLES
SYSVIEWDEP
SYSVIEWS

You can access information in the SQL catalog views by using normal SQL statements. The contents of each of the catalog views is described in the following section.

Catalog Views

The views contained in a SQL catalog are described in this section.

SYSCOLUMNS

The SYSCOLUMNS view contains one row for every column of each table and view in the SQL data base (including the columns of the SQL catalog). The following table describes the columns in the SYSCOLUMNS view:

Column Name	Data Type	Description	
NAME	CHAR(10)	Name of the column	
TBNAME	CHAR(10)	Name of the table or view that contains the column	
TBCREATOR	CHAR(10)	The owner of the table or view	
COLNO	SMALLINT	Numeric place of the column in the table or view, ordered from left to right	
COLTYPE	CHAR(8)	Type of column:	
		INTEGER	Large number
		SMALLINT	Small number
		FLOAT	Floating point; FLOAT, REAL, or DOUBLE PRECISION specified on the CREATE TABLE statement
		CHAR	Fixed-length character string
		DECIMAL	Packed decimal
		NUMERIC	Zoned decimal
LENGTH	SMALLINT	The length attribute of the column; or, in the case of a decimal, numeric, or nonzero precision binary column, its precision:	
		4 bytes	INTEGER
		2 bytes	SMALLINT
		8 bytes	FLOAT, FLOAT(n) where n = 25 to 53, or DOUBLE PRECISION
		4 bytes	FLOAT(n) where n = 1 to 24, or REAL
		Length of string	CHAR
		Precision of number	DECIMAL
		Precision of number	NUMERIC

Column Name	Data Type	Description
SCALE	SMALLINT	Scale of numeric data (zero if not decimal, numeric, or nonzero precision binary)
NULLS	CHAR(1)	If the column can contain null values: N No Y Yes **Note:** This column always contains N.
UPDATES	CHAR(1)	If the column can be changed: N No Y Yes **Note:** The value is N anytime a column cannot be changed.
REMARKS	CHAR(254)	A character string you supply with the COMMENT ON statement
DEFAULT	CHAR(1)	If the column has a default value: N No Y Yes
LABEL	CHAR(30)	A character string you supply with the LABEL ON statement
STORAGE	SMALLINT	The storage requirements for the column: 4 bytes — INTEGER 2 bytes — SMALLINT 8 bytes — FLOAT, FLOAT(n) where n = 25 to 53, or DOUBLE PRECISION 4 bytes — FLOAT(n) where n = 1 to 24, or REAL Length of string — CHAR (Precision/2) + 1 — DECIMAL Precision of number — NUMERIC **Note:** This column supplies the storage requirements for all data types.
PRECISION	SMALLINT	The precision of numeric columns (zero if not numeric) **Note:** This column supplies the precision of all numeric data types, including single- and double-precision floating point.

SYSINDEXES

The SYSINDEXES view contains one row for every index in the SQL data base, including indexes on the SQL catalog. The following table describes the columns in the SYSINDEXES view:

Column Name	Data Type	Description
NAME	CHAR(10)	Name of the index
CREATOR	CHAR(10)	The owner of the index
TBNAME	CHAR(10)	Name of the table on which the index is defined
TBCREATOR	CHAR(10)	The owner of the table
TBDBNAME	CHAR(10)	Name of the SQL data base that contains the table on which the index is defined
UNIQUERULE	CHAR(1)	If the index is unique: D No (duplicates are allowed) U Yes
COLCOUNT	INTEGER	The number of columns in the key
DBNAME	CHAR(10)	Name of the SQL data base that contains the index

SYSKEYS

The SYSKEYS view contains one row for every column of an index type in the SQL data base (including the keys for the indexes on the SQL catalog). The following table describes the columns in the SYSKEYS view:

Column Name	Data Type	Description
IXNAME	CHAR(10)	Name of the index
IXCREATOR	CHAR(10)	The owner of the index
COLNAME	CHAR(10)	Name of the column of the key
COLNO	INTEGER	Numeric position of the column in the row
COLSEQ	INTEGER	Numeric position of the column in the key
ORDERING	CHAR(1)	Order of the column in the key: A Ascending D Descending

SYSTABLES

The SYSTABLES view contains one row for every table or view in the SQL data base (including the columns of the SQL catalog). The following table describes the columns in the SYSTABLES view:

Column Name	Data Type	Description
NAME	CHAR(10)	Name of the table or view
CREATOR	CHAR(10)	The owner of the table or view
TYPE	CHAR(1)	If the row describes a table or view: T Table V View
DBNAME	CHAR(10)	Name of the SQL data base that contains the table or view
COLCOUNT	SMALLINT	Number of columns in the table or view
REMARKS	CHAR(254)	A character string you supply with the COMMENT ON statement
RECLENGTH	SMALLINT	The length of any record in the table
LABEL	CHAR(30)	A character string you supply with the LABEL ON statement

SYSVIEWDEP

The SYSVIEWDEP view records the dependencies of views on tables (including the views of the SQL catalog). The following table describes the columns in the SYSVIEWDEP view:

Column Name	Data Type	Description
BNAME	CHAR(10)	Name of a table the view is dependent on
BCREATOR	CHAR(10)	The owner of the table the view is based on
BDBNAME	CHAR(10)	Name of the SQL data base that contains the table that the view is dependent on
BTYPE	CHAR(1)	Type of object the view was based on: L Logical file P Physical file T Table V View
DNAME	CHAR(10)	Name of the view
DCREATOR	CHAR(10)	The owner of the view

SYSVIEWS

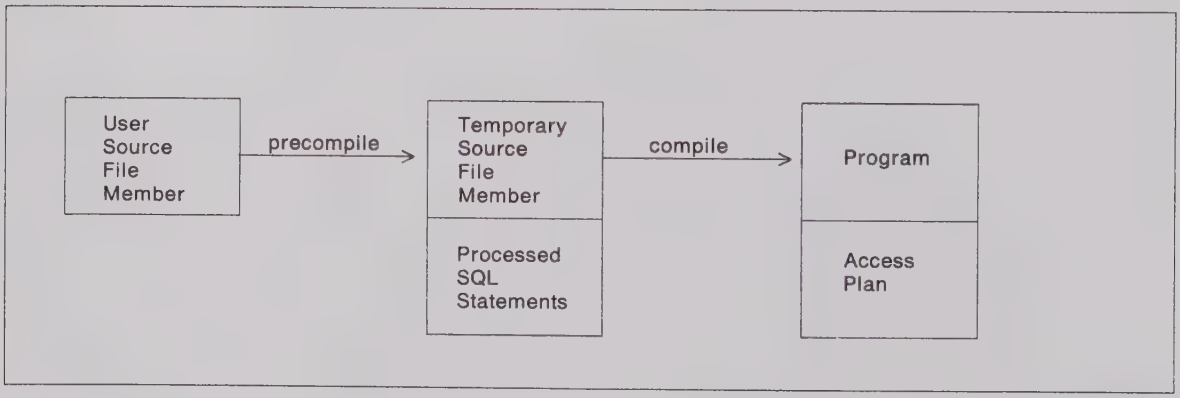
The SYSVIEWS view contains one or more rows for each view in the data base (including the views of the SQL catalog). Each row contains a 70-character portion of the SQL CREATE VIEW statement that created the view. The following table describes the columns in the SYSVIEWS view:

Column Name	Data Type	Description
NAME	CHAR(10)	Name of the view
CREATOR	CHAR(10)	The owner of the view
SEQNO	INTEGER	Sequence number of this row; the first portion of the view is on row one and successive rows have increasing values of SEQNO
CHECK	CHAR(1)	Used only for compatibility with other SQL implementations. For SQL/400, the value is always N.
TEXT	CHAR(70)	The text portion of the text of the CREATE VIEW statement

Compiled Application Program Objects

With SQL/400, you do not have to manage any objects related to the application other than the original source and the resulting program.

The following shows the objects and steps that happen during the precompile and compile processes:



User Source File Member

A source file member contains the programmer's application language and SQL statements. Normally, you create and maintain the source file member by using the source entry utility (SEU), a part of the AS/400 Application Development Tools licensed program.

Temporary Source File Member

The temporary source file member is created in the library (QTEMP) by the precompile process (CRTSQLxxx commands)[2] and is automatically deleted by the system at job completion. A temporary source file member with the same name as the program name is added to QSQLTEMP by the precompile process. This member contains calls to:

- SQL run-time support, which has replaced embedded SQL statements
- Parsed and syntax checked SQL statements

By default, the host language compiler is called by the precompiler. For more information on precompilers, see Chapter 9.

Program

A program is the object created as a result of the compile process, which you can run.

An access plan is a set of internal structures and information that tells SQL how to run an embedded SQL statement in the most effective way. It is created only when a successful compile has occurred. Access plans are stored and maintained in the program object.

---

[2] The xxx in this command refers to the host language indicators CBL for COBOL/400, PLI for AS/400 PL/I, and RPG for RPG III (part of RPG/400).

Chapter 2. Working with SQL Data Bases, Tables, Views, Indexes, and Catalogs

This chapter describes how to create and work with SQL data bases, tables, views, and indexes, and how to use catalogs in data base design. It also describes using the COMMENT ON statement to describe the purpose of a table or view and any other special information about it.

The syntax and parameters for each of the SQL statements used in this chapter are described in detail in the *SQL/400 Reference* manual. The tables referred to in the examples are described in Appendix B.

A detailed description of how to use SQL statements and clauses in more complex situations is provided in Chapter 3.

Creating an SQL Data Base

In SQL/400, the data base is the basic object in which tables, views, and indexes are placed. You must have authority to the CRTLIB and CRTDTADCT CL commands to run the CREATE DATABASE statement. For more information on security, see the *Security Concepts and Planning*.

A sample data base, named USER1, can be created with the following SQL statement:

```
CREATE DATABASE USER1
```

After you have successfully created a data base, you can put tables, views, and indexes in it. For the purpose of supplying sample objects for the examples used in this manual, imagine that we have created the USER1 data base. The following sections describe adding tables and views to the USER1 data base.

Creating and Using a Table

The SQL CREATE TABLE statement is used to create a table and define the physical attributes of the columns in the table.

When this statement is processed, a new, empty table is created, containing the column information as defined. In the CREATE TABLE statement, the first parameter specifies the column name, the second parameter is the data type for that column, and the third parameter specifies if the column can be null. In SQL/400, only NOT NULL or NOT NULL WITH DEFAULT is allowed.

Note: A table must be created in an SQL data base; it cannot be created in a library.

Other than the SQL return code, no other data is ever returned for the CREATE TABLE statement. The full text of return codes and error messages are described in Appendix C.

Creating the Department Table (TDEPT)

The sample department table describes each department in the company and specifies the department manager and the next higher department of authority.

You can create this table with the following interactive SQL statement:

```
CREATE TABLE USER1.TDEPT
    (DEPTNO     CHAR(3)   NOT NULL WITH DEFAULT,
     DEPTNAME   CHAR(36)  NOT NULL WITH DEFAULT,
     MGRNO      CHAR(6)   NOT NULL WITH DEFAULT,
     ADMRDEPT   CHAR(3)   NOT NULL WITH DEFAULT )
```

Using the LABEL ON Statement for Defining Table Names and Column Headings

Sometimes the table name or column name does not clearly define the data when shown on an interactive display of the table. By using the LABEL ON statement, you can create a more descriptive label for the column(s) in addition to, or instead of, the table or column name.

In the interactive environment, the LABEL ON statements looks like this:

```
LABEL ON
  TABLE USER1.TDEPT IS 'Department Structure Table'

LABEL ON
  COLUMN USER1.TDEPT.ADMRDEPT IS 'Reports to Dept.'
```

where the table named TDEPT, when shown on an interactive display, will appear as *Department Structure Table* and the column named *ADMRDEPT* will have the heading *Reports to Dept.* The label for tables can be no more than 30 characters and the label for columns can be no more than 20 characters (blanks included). For more information about the LABEL ON statement, see the *SQL/400 Reference* manual.

Inserting Information into a Table

After you create a table, you can insert information (data values) into it by using the INSERT SQL statement (see "The INSERT Statement" on page 3-1 for more information on using this statement). In the interactive environment, the INSERT statements look like this:

```
INSERT INTO USER1.TDEPT
       (DEPTNO,
        DEPTNAME,
        MGRNO,
        ADMRDEPT )
  VALUES
       ('A00',
        'COMPUTER SERVICE DIV.',
        '000010',
        '   ' )
```

When the INSERT statement is run, a row is placed in the table. Note that a one-to-one correspondence exists between the column names specified in the INSERT clause and the data values specified in the VALUES clause. You must issue an INSERT statement for each row of the table. The sample data base now contains a table filled with data by issuing an INSERT statement for each row.

If COMMIT(*NONE) has been specified, no further action is required. Each insert is done at the time of the operation. However, if you specify COMMIT(*CHG) or COMMIT(*ALL), the insert is not guaranteed until the COMMIT statement is run. Conversely, the insert is backed out if you run a ROLLBACK statement.

Sample Department Table (TDEPT)

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	COMPUTER SERVICE DIV.	000010	(blanks[1])
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER	(blanks[1])	A00
E01	SUPPORT SERVICES	000050	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E11	OPERATIONS	000090	E01
E21	SOFTWARE SUPPORT	000100	E01

[1] These entries were created with blank characters.

Getting Information from a Single Table

You can retrieve stored data by using the SELECT statement. The SELECT statement is the most complex of all SQL statements. This statement is composed of three main clauses:

1. The SELECT clause, which specifies those columns containing the desired data
2. The FROM clause, which specifies the table containing the columns with the desired data
3. The WHERE clause, which supplies a filter that determines which rows of data are retrieved

In addition to the three main clauses, there are several other clauses described in "Using Basic SQL Statements and Clauses" on page 3-1 and in the *SQL/400 Reference* manual, which affect the final form of returned data (for example, the ORDER BY clause).

In the interactive environment, the SQL statement looks like this:

```
SELECT EMPNO, LASTNAME, JOBCODE
   FROM TEMPL
   WHERE JOBCODE > 59
     AND SEX = 'F'
```

Note: The value being compared (in this case F) must be entered exactly as it appears in the table. For example, an f would return an error message. However, in this example, adding the following OR clause would work.

```
AND SEX = 'F' OR SEX = 'f'
```

If the SELECT statement is successful, the result is one or more rows of the specified table.

The running of the above sample SELECT statement returns the following rows from the TEMPL table:

EMPNO	LASTNAME	JOBCODE
000010	HAAS	66
000030	KWAN	60

Note that only those rows are returned whose data values compare with the conditions specified by the predicates in the WHERE clause. Furthermore, the only data values returned are from the columns you explicitly specified in the SELECT clause. Data values of columns other than those explicitly identified are not returned.

Getting Information from More Than One Table

In addition to getting data from one table, SQL allows you to get information from columns contained in different tables. This operation is called a **join** operation. (For a more detailed description of the join operation, see "Joining Data from More Than One Table" on page 3-29). In SQL, a join operation is specified by placing the names of those tables you want to join together into the same FROM clause of a SELECT statement.

For example, consider the following SELECT statement, shown as you would enter it in the interactive environment in the SQL naming convention:

```
SELECT EMPNO, LASTNAME, USER1.TDEPT.DEPTNO, DEPTNAME
   FROM USER1.TEMPL, USER1.TDEPT
   WHERE JOBCODE > 59
     AND SEX = 'F'
     AND USER1.TEMPL.DEPTNO = USER1.TDEPT.DEPTNO
```

Note: Whenever the same column name exists in two or more tables being joined, the table name must be entered before the column name, connected with a period. For example, in the preceding SELECT statement, the column names DEPTNO and EMPNO are found in both tables.

The system naming convention, you must use correlation names to achieve the same results. For example, in the case above, you would enter the following SQL statement:

```
SELECT EMPNO, LASTNAME, Y.DEPTNO, DEPTNAME
   FROM USER1/TEMPL X, USER1/TDEPT Y
   WHERE JOBCODE > 59
     AND SEX = 'F'
     AND X.DEPTNO = Y.DEPTNO
```

where X and Y are correlation names for TEMPL and TDEPT, respectively. For more information on correlation names, see the *SQL/400 Reference* manual.

If the SELECT statement is successful, the result is one or more rows of the specified table. The data values in the columns of the rows returned represent a composite of the data values contained in the two tables specified, TEMPL and TDEPT.

The running of the above example SELECT statement returns the following columns:

EMPNO	LASTNAME	DEPTNO	DEPTNAME
000010	HAAS	A00	COMPUTER SERVICE DIV.
000030	KWAN	C01	INFORMATION CENTER

Changing Information in a Table

Using the UPDATE statement, you can change the data values in some or all columns of a table. Also, you can change more than one row during a single run by using the WHERE clause with UPDATE statement, like in the SELECT statement. If the WHERE clause is absent, all of the rows in the specified table are changed. However, if the WHERE clause is highly restrictive, only one or a few rows are changed. (For more information on using the UPDATE statement and the WHERE clause, see "The UPDATE Statement" on page 3-3 and "The WHERE Clause" on page 3-8.)

For example, consider the following UPDATE statement, shown in the interactive environment, in which Michael L. Thompson's (employee no. 000020) telephone number is changed to 5678:

```
UPDATE TEMPL
   SET PHONENO = '5678'
   WHERE EMPNO = '000020'
```

The TEMPL table, shown in part, would be changed as follows:

EMPNO	FIRSTNME	MIDINIT	LASTNAME	DEPTNO	PHONENO
000010	CHRISTINE	I	HAAS	A00	3978
000020	MICHAEL	L	THOMPSON	B01	5678
000030	SALLY	A	KWAN	C01	4738

If you specify COMMIT(*NONE), no further action is required. Each change is then done at the time of the operation. However, if you specify COMMIT(*CHG) or COMMIT(*ALL), the change is not guaranteed until you run the COMMIT statement. Conversely, the change is backed out if you run a ROLLBACK statement.

Deleting Information in a Table

The DELETE statement allows you to delete entire rows from a table because they no longer contain needed information. The DELETE statement allows you to use the WHERE clause to identify more than one row to be deleted during a single run. (For more information, see "The DELETE Statement" on page 3-4.)

For example, if Eileen W. Henderson left the company, the row containing information about this person must now be deleted. The following DELETE statement, shown in the interactive environment, looks like this:

```
DELETE
   FROM TEMPL
   WHERE EMPNO = '000090'
```

If you specify COMMIT(*NONE), no further action is required. Each delete is then done at the time of the operation. However, if you specify COMMIT(*CHG) or COMMIT(*ALL), the delete is not guaranteed until you run the COMMIT statement. Conversely, the delete is backed out if you run a ROLLBACK statement.

Creating and Using a View

At times, you may find that no single table contains all the information you need, but that the data is scattered among several tables. Also, you may want to get only part of the data contained in a table and not a whole row or column of data, therefore limiting the access to certain kinds of data, such as salary information. For these conditions, SQL lets you create a view.

You create a view in much the same way as you create a table. The view is defined by the CREATE VIEW statement. In order to create views, you must have SELECT authority on the tables on which views are based.

If you do not specify column names for the view, the column names will be the same as those for the table on which the view is based.

Note: A view must be created over tables or files that are in an SQL data base; a view cannot be created over tables or files in a library.

Creating a View on a Single Table

The following example shows how to create a view on a single table. The view is built on a table named USER1.TDEPT, which contains department data. The table has four columns, but the view is only on three of the columns: *DEPTNO*, *DEPTNAME*, and *MGRNO*. The order of the columns in the SELECT clause is the order in which they appear in the view. The CREATE VIEW statement looks like this:

```
CREATE VIEW VDEPT3 AS
  SELECT DEPTNO,DEPTNAME,MGRNO
  FROM USER1.TDEPT
```

In the foregoing example, the columns of the view have the same name as the columns in the table, because no column list follows the view name. The following table is the result of running the SQL statement:

```
SELECT * FROM VDEPT3
```

DEPTNO	DEPTNAME	MGRNO
A00	COMPUTER SERVICE DIV.	000010
B01	PLANNING	000020
C01	INFORMATION CENTER	000030
D01	DEVELOPMENT CENTER	(blanks[2])
E01	SUPPORT SERVICES	000050
D11	MANUFACTURING SYSTEMS	000060
D21	ADMINISTRATION SYSTEMS	000070
E11	OPERATIONS	000090
E21	SOFTWARE SUPPORT	000100

[2] This entry was created with blank characters.

Creating a View Combining Data from More Than One Table

You can create a view that combines data from two or more tables by naming more than one table in the FROM clause. The combining operation is called a **join**. In the following example, the USER1.TEMPL table contains a column of manager's names called *LASTNAME*, and a column of employee serial numbers called *EMPNO*. These are joined with the *MGRNO* column of the USER1.TDEPT table. The CREATE VIEW statement looks like this:

```
CREATE VIEW SMITH.VDEPTM AS
  SELECT USER1.TDEPT.DEPTNO,MGRNO,LASTNAME,ADMRDEPT
  FROM USER1.TDEPT,USER1.TEMPL
  WHERE USER1.TEMPL.EMPNO = USER1.TDEPT.MGRNO
```

The following are the results of running the SQL statement:

```
SELECT * FROM SMITH.VDEPTM
```

DEPT	MGR	NAME	REPORTTO
A00	000010	HAAS	(blanks[3])
B01	000020	THOMPSON	A00
C01	000030	KWAN	A00
E01	000050	GEYER	A00
D11	000060	STEARN	D01
D21	000070	PULASKI	D01
E11	000090	HENDERSON	E01
E21	000100	SPENSER	E01

[3] This entry was created with blank characters.

If you want to create the same view, but including only those departments that report administratively to department A00, and you also want a different set of column names, your CREATE VIEW statement looks like this:

```
CREATE VIEW SMITH.VDEPTMA00
  (DEPT,MGR,NAME,REPORTTO)
  AS
  SELECT USER1.TDEPT.DEPTNO,MGRNO,LASTNAME,ADMRDEPT
  FROM USER1.TEMPL,USER1.TDEPT
  WHERE USER1.TEMPL.EMPNO = USER1.TDEPT.MGRNO
  AND ADMRDEPT = 'A00'
```

The following table is the result of running the SQL statement:

```
SELECT * FROM SMITH.VDEPTMA00
```

DEPT	MGR	NAME	REPORTTO
B01	000020	THOMPSON	A00
C01	000030	KWAN	A00
E01	000050	GEYER	A00

Notes on Using a View

When creating a view, the following restrictions must be considered:

- You cannot change, insert, or delete data in a view if it includes any of the following:
  - A FROM clause that uses more than one table (a join)
  - The FROM clause that identifies a read-only view
  - Any of the SQL column functions (SUM, MAX, MIN, AVG, or COUNT)
  - Elimination of duplicate rows (DISTINCT)
  - Grouping (GROUP BY), or HAVING

  In the above cases, you can get data from the views by means of the SQL SELECT statement, but you cannot use statements such as INSERT, UPDATE, or DELETE.
- You cannot insert a row in a view if:
  - The table on which the view is based has a column for which there is no default value, and that column is not in the view.
  - The view has a column resulting from an expression, a constant, or a function, and the column has been specified in the INSERT column list.
- You cannot update a column of a view that results from an expression, a constant, or a function.
- You cannot use the special register USER.
- You cannot use the scalar function LENGTH.
- You cannot use UNION or UNION ALL.

You can make changes to a table through a view even though the view does not contain the same number of columns or the same number of rows as the table on which it is based, provided that the columns not referred to were defined as NOT NULL WITH DEFAULT.

When you define a view on a table, it is like creating a new table, containing just the columns and rows you want.

You process the view as though it were a table, even though the view is totally dependent on one or more tables for data. The view has no data of its own and hence requires no storage for the data. Because a view is derived from a table that exists in storage, when you update the view data you are really updating data in the table. Therefore, views are automatically kept up to date as the tables they depend on are updated.

Views let you deal only with the data you need. A view reduces complexity and, at the same time, restricts access. When your application uses a view (instead of the table on which the view is based), it cannot access other rows or columns of the table that are not included in the view.

Working with Indexes

An index is used by the system for faster data retrieval. The following example creates an index over the column *LASTNAME* in the USER1.TEMPL table:

```
CREATE INDEX INX1 ON USER1.TEMPL (LASTNAME)
```

Any number of indexes can be created; however, because the indexes are maintained by the system, a large number of indexes can adversely affect performance. See Chapter 13 for more information on using indexes.

Note: An index can be created only over tables or files that are in an SQL data base.

Using the Catalog in Data Base Design

A catalog is automatically created when you create a data base. As the following examples show, you can display catalog information, but you cannot INSERT, DELETE, or UPDATE catalog information.

You must have SELECT privileges on the catalog views to run the following examples.

Getting Catalog Information about a Table

SYSTABLES contains a row for every table and view in the SQL data base. It tells you if the object is a table or view, the object name, the owner of the object, what SQL data base it is in, and so forth.

This sample statement displays information for the USER1.TDEPT table:

```
SELECT *
  FROM USER1.SYSTABLES
  WHERE NAME = 'TDEPT'
```

Getting Catalog Information about a Column

SYSCOLUMNS contains a row for each column of every table and view in the data base.

This sample statement displays all the column names in the USER1.TDEPT table:

```
SELECT *
  FROM USER1.SYSCOLUMNS
  WHERE TBNAME = 'TDEPT'
```

The result of the previous sample statements is a row of information for each column in the table.

For more information about each column, write a SELECT statement like this:

```
SELECT NAME,TBNAME,COLTYPE,LENGTH,DEFAULT
  FROM USER1.SYSCOLUMNS
  WHERE TBNAME = 'TDEPT'
```

In addition to the column name for each column, the SELECT statement shows:

- The name of the table that contains the column
- The data type of the column
- The length attribute of the column
- If the column allows default values

The result looks like this:

NAME	TBNAME	COLTYPE	LENGTH	DEFAULT
DEPTNO	TDEPT	CHAR	3	Y
DEPTNAME	TDEPT	CHAR	36	Y
MGRNO	TDEPT	CHAR	6	Y
ADMRDEPT	TDEPT	CHAR	3	Y

Getting Catalog Information about Indexes

SYSINDEXES contains a row for every index in the data base. The following example gets information about the index INX1:

```
SELECT *
  FROM USER1.SYSINDEXES
  WHERE NAME = 'INX1'
```

The previous example displays a single row of information about a particular index. However, a table can have more than one index. To display information about all the indexes on a table, write a statement like this:

```
SELECT *
  FROM USER1.SYSINDEXES
  WHERE TBNAME = 'TEMPL'
```

Getting Catalog Information about Views

For every view you create in the data base, information is stored in the catalog views. This is what happens in the catalog, following a CREATE VIEW:

- A row is inserted into SYSTABLES.
- A row is inserted into SYSCOLUMNS for each column of the view.
- One or more rows are inserted into SYSVIEWS to record the text of the CREATE VIEW statement.
- For each table on which the view is dependent, a row is inserted into SYSVIEWDEP to record what the view is dependent on.

Building a View over Catalog Views

You can build a view on one or more of the catalog views containing information about your own tables and views. This example creates a view of a user's tables from the SYSTABLES view:

```
CREATE VIEW MYTABLES AS
  SELECT * FROM USER1.SYSTABLES
  WHERE CREATOR = 'USER1'
```

Using COMMENT ON

After you create a table or view, you can supply information about it for future reference, such as the purpose of the table or view, who uses it, and anything unusual or special about it. You can also include similar information about each column of the table or view. Your comment must not be more than 254 bytes.

A comment is especially useful if your names do not clearly indicate the contents of the columns or tables. In that case, use a comment to describe the specific contents of the column or table.

If you include REMARKS in the list of column names you get from SYSCOLUMNS, any comments you had included for the columns are displayed. Both SYSTABLES and SYSCOLUMNS have a column for REMARKS.

An example of using COMMENT ON follows:

```
COMMENT ON TABLE USER1.TEMPL IS
  'Employee table.  Each row in this table represents
   one employee of the company.'
```

Getting Comments

After running a COMMENT ON statement, your comments are stored in the *REMARKS* column of SYSTABLES or SYSCOLUMNS. (If the indicated row had already contained a comment, the old comment is replaced by the new one.) The following example gets the comments added by the COMMENT ON statement in the previous example:

```
SELECT REMARKS
  FROM USER1.SYSTABLES
  WHERE NAME = 'TEMPL'
```

Chapter 3. SQL Coding Techniques

This chapter describes coding techniques for:

- Using basic SQL statements and clauses
- Using the UNION keyword to combine SELECT statements
- Using a cursor
- Performing complex search conditions
- Joining data from more than one table
- Inserting multiple rows into a table
- Inserting default values into columns

You can embed SQL statements in an application program or issue them interactively, using interactive SQL (described in Chapter 10).

Using Basic SQL Statements and Clauses

This section shows the basic SQL statements and clauses that retrieve, update, delete, and insert data into tables and views. The SQL statements used are SELECT, UPDATE, DELETE, and INSERT. Also, FETCH can be used in an application program to access data. Examples using these SQL statements are supplied to help you develop SQL statements. Detailed syntax and parameter descriptions for SQL statements are given in the *SQL/400 Reference* manual.

You can write SQL statements on one line or on many lines. The rules for the continuation of lines are the same as those of the host language (the language the program is written in).

Notes:

1. The SQL statements described in this chapter can be run on both SQL created and nonSQL created tables and views. The tables and views can be either in an SQL data base or in a library.
2. Character strings specified in an SQL statement (such as used with WHERE or VALUES clauses) are case sensitive; that is, uppercase characters must be entered in uppercase and lowercase characters must be entered in lowercase.

The INSERT Statement

You can use the INSERT statement to add new rows to a table or view by doing one of the following:

- Specifying values in the INSERT statement for columns of the single row to be added.
- Including a SELECT statement in the INSERT statement to tell SQL what data for the new row is contained in another table or view. "Inserting Multiple Rows into a Table" on page 3-34, explains how to use the SELECT statement within an INSERT statement to add multiple rows to a table.

Note: Because views are built on tables and actually contain no data, working with views can be complicated. See "Notes on Using a View" on page 2-10 for more information on inserting information into a view.

In any case, for every row you insert, you must supply a value for each NOT NULL column of a table that does not have a default value. The INSERT statement for adding a row to a table or view may look like this:

```
EXEC SQL
  INSERT INTO table-name
    [(column-name-1 [, column-name-2] ... )]
    VALUES (value-1 [, value-2] ... )
END-EXEC.
```

The INTO clause names the columns for which you specify values. The VALUES clause specifies a value for each column named in the INTO clause.

For example, suppose you want to add a new employee to the USER1.TEMPL table:

```
EXEC SQL
  INSERT INTO USER1.TEMPL
    (EMPNO, FIRSTNME, MIDINIT, LASTNAME, DEPTNO)
    VALUES (:PGM-SERIAL, :PGM-NAME1,
      :PGM-NAME2, :PGM-NAME3, :PGM-DEPT)
END-EXEC.
```

Initially, you might supply values for only those columns that have no default values. SQL inserts the new values (contained in host variables) into the USER1.TEMPL table in the order in which you list them. The first value (from PGM-SERIAL) is put into the first specified column (*EMPNO*), the second value (from PGM-NAME1) is put into the second specified column (*FIRSTNME*), and so on. If you specify fewer column names than there are columns in the row, SQL places default values in the remaining columns.

You must provide a value in the VALUES clause for each column named in an INSERT statement's column list. You can name all columns for which you are providing values. Alternatively, you can omit the column name list.

It is a good idea to name all columns into which you are inserting values because:

Your source statements are more descriptive.
You can verify that you are giving the values in order.
You have better data independence.

When your program attempts to insert a row that duplicates another row already in the table, an error might occur:

- If the table has a unique index, the row is not inserted. Instead, SQL returns an SQLCODE of −803.
- If the table does not have a unique index, the row can be inserted without error.

If SQL finds an error while running the INSERT statement, it stops inserting data. No rows in the table are inserted (rows already inserted, if any, are deleted), if COMMIT(*ALL) or COMMIT(*CHG) is specified. If you specify COMMIT(*NONE), any rows already inserted are *not* deleted.

If the row is inserted without error, the SQLERRD(3) field of the SQLCA will have a value of 1.

Note: INSERT with subselect may have more than one row to insert. The number of rows inserted is reflected in SQLERRD(3).

The UPDATE Statement

To change the data in a table, use the UPDATE statement. With the UPDATE statement, you can change the value of one or more columns in each row that satisfies the search condition of the WHERE clause. The result of the UPDATE statement is one or more changed column values in zero or more rows of a table (depending on how many rows satisfy the search condition specified in the WHERE clause). The UPDATE statement looks like this:

```
EXEC SQL
  UPDATE table-name
    SET column-name-1 = value-expression
    [, column-name-2 = value-expression ] ...
    WHERE search-condition ...
END-EXEC.
```

For example, suppose an employee has been relocated. To update several items of the employee's data in the USER1.TEMPL table to reflect the move, you can specify:

```
EXEC SQL
  UPDATE USER1.TEMPL
    SET JOBCODE = :PGM-CODE,
    PHONENO = :PGM-PHONE
    WHERE EMPNO = :PGM-SERIAL
END-EXEC.
```

First, name the table or view, then provide a `column-name = value-expression` pair for each column you want to update. Use the SET clause to specify the new value for a column. The SET clause names the columns you want updated and provides the values you want them changed to. The value-expression you specify can be:

A **column name.** Replace the column's current value with the contents of another column in the same row.

A **constant.** Replace the column's current value with the value provided in the SET clause.

A **host variable.** Replace the column's current value with the contents of the host variable.

A **special register.** Replace the column's current value with a special register value (USER).

An **expression.** Replace the column's current value with the value that results from an expression.

Next, identify the row(s) to be updated:

- To update a single row, use a WHERE clause that locates one, and only one, row.
- To update several rows, use a WHERE clause that locates only the rows you want to update.

You can omit the WHERE clause; if you do, SQL updates each row in the table or view with the values you supply.

If SQL finds an error while running your UPDATE statement, it stops updating and returns a negative SQLCODE. If you specify COMMIT(*ALL) or COMMIT(*CHG), no rows in the table are changed (rows already changed, if any, are restored to their previous values). If COMMIT(*NONE) is specified, any rows already changed are *not* restored to previous values.

If SQL cannot find any rows that satisfy the search condition, a SQLCODE of +100 is returned.

Note: UPDATE with subselect may have more than one row to update. The number of rows updated is reflected in SQLERRD(3).

The DELETE Statement

To remove rows from a table, use the DELETE statement. When you DELETE a row, you remove the entire row. DELETE does not remove specific columns from the row. The result of the DELETE statement is the removal of zero or more rows of a table (depending on how many rows satisfy the search condition specified in the WHERE clause). If you omit a WHERE clause from a DELETE statement, SQL removes all the rows of the table. The DELETE statement looks like this:

```
DELETE FROM table-name
  WHERE search-condition ...
```

For example, suppose department D11 was moved to another place. You want to delete each row in the USER1.TEMPL table with a DEPTNO value of D11.

```
EXEC SQL
  DELETE FROM USER1.TEMPL
    WHERE DEPTNO = 'D11'
END-EXEC.
```

When this statement is run, SQL deletes any row from the USER1.TEMPL table that meets the search condition.

The WHERE clause tells SQL which rows you want to delete from the table. SQL deletes all the rows that satisfy the search condition from the base table. You can omit the WHERE clause, but you will probably want to include one, because a DELETE statement without a WHERE clause deletes all the specified rows from the table or view. To delete a table definition as well as the table contents, issue the DROP statement (described in the *SQL/400 Reference* manual).

If SQL finds an error while running your DELETE statement, it stops deleting data and returns a negative SQLCODE. If you specify COMMIT(*ALL) or COMMIT(*CHG), no rows in the table are deleted (rows already deleted, if any, are restored to their previous values). If COMMIT(*NONE) is specified, any rows already deleted are *not* restored to their previous values.

If SQL cannot find any rows that satisfy the search condition, then a SQLCODE of +100 is returned.

Note: DELETE with subselect may have more than one row to delete. The number of rows deleted is reflected in SQLERRD(3).

The SELECT INTO Statement

You can use the SELECT INTO statement[1] to retrieve a specific row (for example, the row for an employee). The format and syntax shown here are very basic. SELECT statements can be much more varied than the examples presented in this chapter. A SELECT INTO statement specifies six items of information:

1. The name of each column you want
2. The name of each host variable used to contain retrieved data
3. The name of the table or view that contains the data
4. A search condition to uniquely identify the row that contains the information you want
5. The name of each column used to group your data
6. A search condition that uniquely identifies a group that contains the information you want

A SELECT INTO statement looks like this:

```
EXEC SQL
  SELECT column names
    INTO host variables
    FROM table or view name
    WHERE search condition
    GROUP BY column names
    HAVING search conditions
END-EXEC.
```

The INTO clause names the host variables (variables in your program used to contain retrieved column values). The value of the first column specified in the SELECT clause is put into the first host variable named in the INTO clause; the second value is put into the second host variable; and so on.

The INTO clause is part of the SELECT statement because we are assuming the WHERE clause can be satisfied by only one row. For example, each row in the USER1.TEMPL table has a unique EMPNO (employee number). Therefore, the result of the SELECT statement is assumed to be either one or zero rows. Finding more than one row is an error, and no data is returned.

If you want several rows to be the result of a SELECT statement, use a DECLARE CURSOR statement to select the rows, followed by a FETCH statement to move the column values into host variables one row at a time. A technique for using cursors is described in "Using a Cursor" on page 3-20.

The FROM clause names the table (or view) that contains the data you are interested in.

---

[1] For the complete syntax of the SELECT statement, see the *SQL/400 Reference* manual.

For example, assume that each department listed in the USER1.TDEPT table has a unique department number. You want to retrieve the department name and manager number from the USER1.TDEPT table for department C01. To do this, your program can issue:

```
EXEC SQL
  SELECT DEPTNAME, MGRNO
    INTO :PGM-DEPTNAME, :PGM-MGRNO
    FROM USER1.TDEPT
    WHERE DEPTNO = :PGM-DEPT
END-EXEC.
```

In this example, PGM-DEPT has been set to C01. When the statement is run, the result is one row:

PGM-DEPTNAME	PGM-MGRNO
INFORMATION CENTER	000030

If SQL is unable to find a row that satisfies the search condition, a SQLCODE of +100 is returned.

If SQL finds an arithmetic expression error while running your statement, one of two things occurs:

- If the arithmetic expression error occurs in the SELECT list or during the evaluation of an argument of a column or scalar function in the SELECT list, and an indicator variable is provided for the expression in error:
  - SQL returns a −2 for the indicator variable corresponding for the expression in error.
  - SQL returns all valid data for that row.
  - SQL returns an SQLCODE of +802.
- If the arithmetic expression error occurs under any other circumstances, SQL stops retrieving rows and returns an SQLCODE of −802 in the SQLCA.

If SQL finds a numeric conversion error while running your SELECT statement, one of two things occurs:

- If the numeric conversion error occurs while converting a numeric value into the host variable and an indicator variable is provided with the host variable:
  - SQL returns a −2 for the host variable corresponding for the expression in error.
  - SQL returns all valid data for that row.
  - SQL returns an SQLCODE of +304.
- If the numeric conversion error occurs while converting a numeric value into the data type of the host variable and an indicator variable is not provided with the host variable, SQL stops running your statement and returns an SQLCODE of −304.

In the case of SQLCODEs of −304 or −802, the SQLCA reports only the first conversion error or arithmetic error detected. However, the indicator variable corresponding to each results column, having arithmetic and/or conversion errors, will be set to −2.

If SQL finds any other errors while running your SELECT statement, a negative SQLCODE is returned.

If SQL finds that the retrieved character column value length is too large to be saved in a host variable, SQL:

- Truncates the data while assigning the value to the host variable.
- Sets SQLWARN0 and SQLWARN1 in the SQLCA to the value W.
- Sets the indicator variable, if provided, to the length of the value before truncation.

Specifying the Columns You Want

With the SELECT clause (the first part of a SELECT statement), you specify the name of each column you want to retrieve. For example:

```
SELECT EMPNO, LASTNAME, DEPTNO
⋮
```

You can specify that only one column be retrieved, or as many as 8000 columns. The value of each column you name is retrieved in the order specified in the SELECT clause.

If you want to retrieve all columns (in the same order as they appear in the row), use an asterisk (*) instead of naming the columns:

```
SELECT *
⋮
```

You can issue a SELECT * statement from a program, but this is not recommended because, if the table definition changes, your program might reference columns for which no receiving host variables are defined.

When using the SELECT statement in an application program, list the column names to give your program more data independence. There are two reasons for this:

1. When you look at the source code statement, you can easily see the one-to-one correspondence between the column names in the SELECT clause and the host variables named in the INTO clause.
2. If a column is added to a table or view you access and you use "SELECT * ...," the INTO clause does not have a matching host variable named for the new column. The extra column causes you to get a warning (not an error) in the SQLCA.

Processing Data in a View

You can retrieve data from a view in exactly the same way you retrieve data from a table. However, there are several restrictions when you attempt to update, insert, or delete data in a view. These restrictions are described in "Notes on Using a View" on page 2-10.

The WHERE Clause

The WHERE clause specifies a search condition that identifies the row or rows you want to retrieve, update, or delete. The number of rows you process with an SQL statement, then, depends on the number of rows that satisfy the WHERE clause **search condition**. A search condition consists of one or more **predicates**. A predicate specifies a test that you want SQL to apply to a specified row(s) of a table.

In the following example, DEPTNO = 'C01' is a predicate, DEPTNO and 'C01' are expressions, and the equal sign (=) is a comparison operator. Note that character values are enclosed in apostrophes ('); numeric values are not. This applies to all literal values wherever they are coded within an SQL statement. For example, to specify that you are interested in the rows where the department number is C01, you would say:

```
...
WHERE DEPTNO = 'C01'
```

In this case, the search condition consists of one predicate: DEPTNO = 'C01'.

Using Expressions in the WHERE Clause

An expression in a WHERE clause names or specifies something you want to compare to something else. Each expression, when resolved by SQL, has a character string or a numeric value. The expressions you specify can be:

- A **column name** names a column. For example:

  ```
  ...
  WHERE EMPNO = '000200'
  ```

  EMPNO names a column that is defined as a 6-byte character value. Equality comparisons (that is, X = Y or X ¬= Y) can be performed on character data. Other types of comparisons can also be evaluated for character data, based on the binary collating sequence.

  However, you cannot compare character strings to numbers. You also cannot perform arithmetic operations on character data (even though EMPNO is a character string that appears to be a number).

- An **expression** identifies two values that are added (+), subtracted (−), multiplied (*), divided (/), or concatenated (||) to result in a value. The operands of an expression can be:

  A constant (that is, a literal value)
  A column
  A host variable
  A value returned from a function
  A special register
  Another expression

  For example:

  ```
  ...
  WHERE JOBCODE + EDUCLVL > 70
  ```

JOBCODE names a column that is defined as a 3-digit packed decimal value (DECIMAL(3)). *EDUCLVL* names a column that is defined as a halfword integer value (SMALLINT).

- A **constant** specifies a literal value for the expression. For example:

```
...
WHERE 40000 < SALARY
```

SALARY names a column that is defined as an 8-digit packed decimal value (123456.78).

- A **host variable** identifies a variable in an application program. For example:

```
...
WHERE EMPNO = :EMP
```

- A **special register** identifies a special value generated by the data base manager. For example:

```
...
WHERE LASTNAME = USER
```

A search condition need not be limited to two column names or constants separated by an arithmetic or comparison operator. You can develop a complex search condition that specifies several predicates separated by AND and OR, names and constants. No matter how complex the search condition, it will supply a TRUE or FALSE value when evaluated against a row. There is also an *unknown* truth value, which is effectively false. That is, if the value of a row is null, this null value is not returned as a result of a search because it is not less than, equal to, or greater than the value specified in the search condition. More complex search conditions and predicates are described in "Performing Complex Search Conditions" on page 3-26.

To fully understand the WHERE clause, you need to know how SQL evaluates search conditions and predicates, and compares the values of expressions. This topic is discussed in the *SQL/400 Reference* manual.

Comparison Operators: The SQL comparison operators are:

=	Equal to
¬= or <>	Not equal to
<	Less than
>	Greater than
<= or ¬>	Less than or equal to (or not greater than)
> = or ¬<	Greater than or equal to (or not less than)

The NOT Keyword

You can precede a predicate with the NOT keyword to specify that you want the opposite of the predicate's value (that is, TRUE if the predicate results are FALSE, or vice versa). For example, to indicate that you are interested in all employees except those working in department C01, you could say:

```
...
WHERE NOT DEPTNO = 'C01'
```

which is equivalent to:

```
...
WHERE DEPTNO ¬= 'C01'
```

Note: You cannot make the NOT keyword part of a comparison operator: NOT must precede a predicate.

The USER Special Register

When an SQL statement containing the USER special register is run, SQL replaces USER with the authorization ID of the person running the program.

There are two restrictions on using the USER special register:

1. USER cannot be used in a CREATE VIEW statement.
2. The authorization ID of the person running the program cannot be longer than 8 characters.

Multiple Predicates

A WHERE clause can contain several predicates, each of which is separated by a **logical connective**, AND or OR. For example, to locate all male employees in department D21, you can specify:

```
...
WHERE SEX = 'M' AND DEPTNO = 'D21'
```

You can precede a predicate with the NOT keyword to specify the opposite of the predicate. However, NOT applies only to the predicate it precedes, not to all predicates in the WHERE clause. For example, to specify all women who work in department D11:

```
...
WHERE NOT SEX = 'M'
      AND DEPTNO = 'D11'
```

The GROUP BY Clause

Without a GROUP BY clause, the application of SQL column functions yields *one* row. When GROUP BY is used, the function is applied to *each* group, thereby yielding as many rows as there are groups.

The GROUP BY clause allows you to find the characteristics of groups of rows rather than individual rows. When you specify a GROUP BY clause, SQL divides the selected rows into groups such that the rows of each group have matching values in one or more columns. Next, SQL processes each group to produce a single-row result for the group. You can specify one or more columns in the GROUP BY clause to group the rows. The items you specify in the SELECT statement are properties of each group of rows, not properties of individual rows in a table or view.

For example, the USER1.TEMPL table has several sets of rows, and each set consists of rows describing members of a specific department. To find the average salary of people in each department, you could issue:

The SQL statement:	Results in:

EXEC SQL
  DECLARE XMP1 CURSOR FOR
  SELECT DEPTNO,
  DECIMAL(AVG(SALARY),5,0)
  FROM USER1.TEMPL
  GROUP BY DEPTNO
END-EXEC.
...
EXEC SQL
  FETCH XMP1
  INTO :WORK-DEPT, :AVG-SALARY
END-EXEC.
	fetch
	1→ A00 42833
	2→ B01 41250
	... ...
	WORK-DEPT AVG-SALARY

The result is several rows, one for each department.

Note: Grouping the rows does not mean ordering them. Grouping puts each selected row in a group, which SQL then processes to derive characteristics of the group. Ordering the rows puts all the rows in the results table in ascending or descending collating sequence. (The "The ORDER BY Clause" on page 3-14 describes how to do this.)

When you use GROUP BY, you name the column(s) you want SQL to use to group the rows. For example, suppose you want a list of the number of people working on each major project described in the USER1.TPROJ table. You could issue:

The SQL statement:	Results in:

EXEC SQL
  DECLARE XMP2 CURSOR FOR
  SELECT SUM(PRSTAFF), MAJPROJ
  FROM USER1.TPROJ
  GROUP BY MAJPROJ
END-EXEC.
...
EXEC SQL
  FETCH XMP2
  INTO :SUM-PR, :MAJ-PROJ
END-EXEC.
	fetch
	1→ 3 MA2100
	2→ 6 MA2110
	3→ 6 AD3100
	... ...
	SUM-PR MAJ-PROJ

The result is a list of the company's current major projects and the number of people working on each project.

You can also specify that you want the rows grouped by more than one column. For example, you could issue a SELECT statement to find the average salary for men and women in each department, using the USER1.TEMPL table. To do this, specify:

The SQL statement:

```
EXEC SQL
  DECLARE XMP3 CURSOR FOR
    SELECT DEPTNO, SEX,
        DECIMAL(AVG(SALARY),5,0)
      FROM USER1.TEMPL
      GROUP BY DEPTNO, SEX
END-EXEC.
 ...
EXEC SQL
  FETCH XMP3
    INTO :DEPT, :SEX,
      :AVG-WAGES
END-EXEC.
```

Results in:

	DEPT	SEX	AVG-WAGES
fetch 1→	A00	F	52750
2→	A00	M	37875
3→	B01	M	41250
4→	C01	F	30156
	...	...	...

Because you did not include a WHERE clause in this example, SQL examines and process all rows in the USER1.TEMPL table. The rows are grouped first by department number and next (within each department) by sex before SQL derives the average SALARY value for each group.

The HAVING Clause

You can use the HAVING clause to specify a search condition for the groups selected based on a GROUP BY clause. HAVING says that you want *only* those groups that satisfy the condition in the HAVING clause. Therefore, the search condition you specify in the HAVING clause must test properties of each group rather than properties of individual rows in the group.

The HAVING clause follows the GROUP BY clause and can contain the same kind of search condition you can specify in a WHERE clause. In addition, you can specify column functions in a HAVING clause. For example, suppose you wanted to retrieve the average salary of women in each department. To do this, you would use the AVG column function and group the resulting rows by DEPTNO and specify a WHERE clause of SEX = 'F'.

To specify the further condition that you want this data only when all the employees in the selected department who have an education level equal to or greater than 16 (a college graduate), use the HAVING clause. The HAVING clause tests a property of the group. In this case, the test is on MIN(EDUCLVL), which is a group property:

The SQL statement:

```
EXEC SQL
  DECLARE XMP4 CURSOR FOR
    SELECT DEPTNO,
        DECIMAL(AVG(SALARY),5,0),
        MIN(EDUCLVL)
      FROM USER1.TEMPL
      WHERE SEX = 'F'
      GROUP BY DEPTNO
      HAVING MIN(EDUCLVL) >= 16
END EXEC.
 ...
EXEC SQL
  FETCH XMP4
    INTO :DEPT, :AVG-WAGES,
      :MIN-EDUC
END-EXEC.
```

Results in:

fetch	DEPT	AVG-WAGES	MIN-EDUC
1→	A00	52750	18
2→	C01	30156	16
3→	D11	24476	17
...	...	...	...

You can use multiple predicates in a HAVING clause by connecting them with AND and OR, and you can use HAVING NOT for any predicate of a search condition.

The ORDER BY Clause

You can specify that you want selected rows retrieved in a particular order, sorted by ascending or descending collating sequence of a column's value, with the ORDER BY clause. You can use an ORDER BY clause as you would a GROUP BY clause: specify the name of the column(s) you want SQL to use when retrieving the rows in a collated sequence.

Instead of naming the columns to order the results, you can use a number. For example, ORDER BY 3 specifies you want the results ordered by the *third* column of the results table, as specified by the SELECT statement. Use a number to order the rows of the results table when the sequencing value is the result of an expression, column function, or something other than a column name in the SELECT statement.

For example, to retrieve the names and department numbers of female employees listed in the alphanumeric order of their department numbers, you could use this SELECT statement:

The SQL statement:

```
EXEC SQL
  DECLARE XMP5 CURSOR FOR
    SELECT LASTNAME, DEPTNO
      FROM USER1.TEMPL
      WHERE SEX = 'F'
      ORDER BY DEPTNO
END-EXEC.
 ...
EXEC SQL
  FETCH XMP5
    INTO :PGM-NAME3, :DEPT
END-EXEC.
```

Results in:

fetch	PGM-NAME3	DEPT
1→	HAAS	A00
2→	KWAN	C01
3→	QUINTANA	C01
4→	NICHOLLS	C01
5→	PIANKA	D11
6→	SCOUTTEN	D11
7→	LUTZ	D11
8→	PULASKI	D21
9→	JOHNSON	D21
10→	PEREZ	D21
11→	HENDERSON	E11
12→	SCHNEIDER	E11

Note: All columns named in the ORDER BY clause must also be named in the SELECT statement.

You can also specify whether you want SQL to collate the rows in ascending (ASC) or descending (DESC) sequence. An ascending collating sequence is the default. In the above SELECT statement, SQL first returns the row with the lowest department number (alphabetically and numerically), followed by rows with higher department numbers. To specify that you want the rows ordered in descending collating sequence based on the department number, you say:

```
...
 ORDER BY DEPTNO DESC.
```

As with GROUP BY, you can specify a secondary ordering sequence (or several levels of ordering sequences) as well as a primary one. In the example above, you might want the rows ordered first by department number, and, within each department, ordered by employee name. To do this, specify:

```
...
ORDER BY DEPTNO, LASTNAME.
```

Note: If you intend to update a column or delete a row, you cannot include a GROUP BY or HAVING clause in the SELECT statement within a DECLARE CURSOR statement. (The DECLARE CURSOR statement is described in "Using a Cursor" on page 3-20.)

Using the UNION Keyword to Combine SELECT Statements

Using the UNION keyword, you can combine two or more SELECT statements to form a single SELECT statement. When SQL encounters the UNION keyword, it processes each SELECT statement to form an interim results table, then it combines the interim results table of each statement and deletes duplicate rows to form a combined results table. You use UNION to merge lists of values from two or more tables. You can use any of the clauses and techniques you have learned so far when coding SELECT statements, including ORDER BY.

You can use UNION to eliminate duplicates when merging lists of values obtained from several tables. For example, you can obtain a combined list of employee numbers that includes:

- People in department D11
- People whose assignments include projects MA2112, MA2113, and AD3111.

The combined list is derived from two tables and contains no duplicates. To do this, code:

```
MOVE 'D11' TO WORK-DEPT.
 ...
  EXEC SQL
  DECLARE XMP6 CURSOR FOR
  SELECT EMPNO
    FROM USER1.TEMPL
    WHERE DEPTNO = :WORK-DEPT
  UNION
  SELECT EMPNO
    FROM USER1.TEMPRACT
    WHERE PROJNO = 'MA2112' OR
          PROJNO = 'MA2113' OR
          PROJNO = 'AD3111'
  ORDER BY 1
END-EXEC.
 ...
EXEC SQL
  FETCH XMP6
    INTO :EMP-NUMBER
END-EXEC.
```

To better understand what results from these SQL statements, imagine that SQL goes through the following process:

Step 1: SQL processes the first SELECT statement:	Which results in an interim results table:
... SELECT EMPNO FROM USER1.TEMPL WHERE DEPTNO = 'D11' ...	(from USER1.TEMPL) 000060 000150 000160 000170 ...

Step 2: SQL processes the second SELECT statement:	Which results in another interim results table:
... SELECT EMPNO FROM USER1.TEMPRACT WHERE PROJNO = 'MA2112' OR PROJNO = 'MA2113' OR PROJNO = 'AD3111' ...	(from USER1.TEMPRACT) 000170 000190 000180 ...

Step 3. SQL combines the two interim results tables:	Which results in a combined results table with values in ascending sequence:
EXEC SQL DECLARE XMP6 CURSOR FOR SELECT ... UNION SELECT ... ORDER BY 1 END-EXEC. ... EXEC SQL FETCH XMP6 INTO :EMP-NUMBER END-EXEC.	EMP-NUMBER fetch 1→ 000060 2→ 000150 3→ 000160 4→ 000170 5→ 000180 ... ...

When you use UNION:

- Any ORDER BY clause must appear after the last SELECT statement that is part of the union. In this example, the results are sequenced on the basis of the first selected column, *EMPNO*. The ORDER BY clause specifies that the combined results table is to be in collated sequence.

  Note: To specify the columns that SQL should order the results by, use numbers (in a union, you cannot use column names for this). The number refers

to the position of the column name in the list of columns you include in your SELECT statement.

- You cannot use UNION when creating a view.

To identify which SELECT statement each row is from, you can include a constant at the end of the select list of each SELECT statement in the union. When SQL returns your results, the last column contains the constant for the SELECT statement that is the source of that row. For example, you can code:

```
SELECT A, B, 'A1' ... UNION SELECT X, Y, 'B2'
```

When a row is presented to your program, it includes a value (either A1 or B2) to indicate the table that is the source of the row's values. If the column names in the union are different, SQL uses the first set of column names specified in the SELECT statement when interactive SQL displays or prints the results, or in the SQLDA resulting from processing an SQL DESCRIBE statement.

In the above example, SQL uses A and B.

Restrictions for the Length and Data Type of Columns

When you use UNION, the lengths and data types of the columns named in the SELECT statements must be comparable; any necessary conversions are made to produce the values of the results table. For example, if you code:

```
SELECT EMPNO ... UNION SELECT DEPTNO ...
```

where *EMPNO* is CHAR(6) and *DEPTNO* is CHAR(3), the column in the results table is CHAR(6). The values in the results table that are derived from *DEPTNO* are padded on the right with blanks.

Rules for Numeric Columns

If you want to combine two numeric columns, A and B, the following rules apply:

- If column A or column B is floating-point, the results column is floating-point. If either column A or column B is double-precision floating-point, the results column is double-precision floating-point. If column A and column B are single-precision floating-point, the results column is single-precision floating-point. If either column A or column B is single-precision floating-point and the other is decimal, zoned decimal (NUMERIC), or binary integer, the results column is double-precision floating-point.
- If column A and column B are decimal, or one is decimal and the other binary integer or zoned decimal (NUMERIC), the results column is decimal.
- If column A and column B are zoned decimal (NUMERIC), or one is zoned decimal and the other binary integer, the results column is zoned decimal (NUMERIC).
- If column A or column B are large integer, the results column is large integer. If column A and column B are small integer, the results column is small integer.
- If column A and column B are nonzero scale binary and have different scales, or if one is nonzero scale binary and the other is zero scale binary integer, the results column is decimal.
- If column A and column B are nonzero scale binary and have the same scales, the results column is nonzero scale binary with the scale of A and B.

Rules for Character String Columns

If you want to combine two string columns, A and B, the following rules apply:

- Column A and column B must both be character string.
- If column A and column B are fixed length, the results column is fixed length. Otherwise, the results column is varying-length. The length attribute of the results column is the greater of the length attributes of column A and column B.

Specifying UNION ALL

If you want to keep duplicates in the result of a UNION, specify UNION ALL instead of just UNION.

Step 3. SQL combines two interim results tables:	Resulting in a results table that includes duplicates:

EXEC SQL
  DECLARE XMP6 CURSOR FOR
  SELECT ...
  UNION ALL
  SELECT ...
  ORDER BY 1
END-EXEC.
...
EXEC SQL
  FETCH XMP6
  INTO :EMP-NUMBER
END-EXEC.
	EMP-NUMBER
	fetch
	1 → 000060
	2 → 000150
	3 → 000160
	4 → 000170
	5 → 000170
	6 → 000180
	7 → 000180
	8 → 000190
	... ...

- The UNION ALL operation is associative, for example:

```
(SELECT PROJNO FROM USER1.TPROJ
UNION ALL
SELECT PROJNO FROM USER1.TPROJEC)
UNION ALL
SELECT PROJNO FROM USER1.TEMPRACT
```

  gives the same result as:

```
SELECT PROJNO FROM USER1.TPROJ
UNION ALL
(SELECT PROJNO FROM USER1.TPROJEC
UNION ALL
SELECT PROJNO FROM USER1.TEMPRACT)
```

- When you include the UNION ALL in the same SQL statement as a UNION operator, however, the result of the operation depends on the order of evaluation. Where there are no parentheses, evaluation is from left to right. Where parentheses are included, the parenthesized subselect is evaluated first, followed, from left to right, by the other parts of the statement.

Using a Cursor

Assume that SQL builds a results table[2] to hold all the rows retrieved by running the SELECT statement. SQL then uses a **cursor** to make rows from the results table available to your program. A cursor identifies the current row of the results table specified by a SELECT statement. When you use a cursor, your program can retrieve each row sequentially from the results table until end-of-data (SQLCODE = +100) is reached. The set of rows obtained as a result of running the SELECT statement can consist of zero, one, or many rows, depending on the number of rows that satisfy the search condition.

Note: The SELECT statement referred to in this section must be within a DECLARE CURSOR statement, and cannot include an INTO clause. The DECLARE CURSOR statement defines and names the cursor and specifies the set of rows to be retrieved with the embedded SELECT statement.

The results table of a cursor is processed much like a sequential data set. The cursor must be opened (with an OPEN statement) before any rows are retrieved. A FETCH statement is used to retrieve the cursor's current row. FETCH can be run repeatedly until all rows have been retrieved. When the end-of-data condition occurs, you should close the cursor with a CLOSE statement (similar to end-of-file processing).

Your program can have several cursors; each cursor requires its own:

DECLARE CURSOR statement to define the cursor
OPEN and CLOSE statements to open and close the cursor
FETCH statement to retrieve rows from the cursor's results table

Example of How to Use a Cursor

Suppose your program examines data about people in department D11. The data is kept in the USER1.TEMPL table. The following example shows the SQL statements you would include in a program to define and use a cursor. In this example, the cursor is used by the program to process a set of rows from the USER1.TEMPL table.

[2] SQL uses the concept of results table different ways, depending on the complexity of the SELECT statement. However, the concept is the same.

SQL Statement	Described in Section

EXEC SQL
  DECLARE THISEMP CURSOR FOR
  SELECT EMPNO, LASTNAME,
  DEPTNO, JOBCODE
  FROM USER1.TEMPL
  WHERE DEPTNO = 'D11'
  FOR UPDATE OF JOBCODE
END-EXEC.
	"Step 1: Define the Cursor" on page 3-22.

EXEC SQL OPEN THISEMP END-EXEC.	"Step 2: Open the Cursor" on page 3-23.

EXEC SQL
  WHENEVER NOT FOUND
  GO TO CLOSE-THISEMP
END-EXEC.
	"Step 3: Specify What to Do when End-of-Data Is Reached" on page 3-23.

EXEC SQL
  FETCH THISEMP
  INTO :EMP-NUM, :NAME2,
  :DEPT, :JOB-CODE
END-EXEC.
	"Step 4: Retrieve a Row Using the Cursor" on page 3-23.

... for specific employees
in department D11, update
the JOBCODE value:
 	
EXEC SQL
  UPDATE USER1.TEMPL
  SET JOBCODE = :NEW-CODE
  WHERE CURRENT OF THISEMP
END-EXEC.
 	
... then print the row.
	"Step 5a: Update the Current Row" on page 3-24.

... for other employees,
delete the row:
 	
EXEC SQL
  DELETE FROM USER1.TEMPL
  WHERE CURRENT OF THISEMP
END-EXEC.
	"Step 5b: Delete the Current Row" on page 3-24.

Branch back to fetch and process the next row.	
CLOSE-THISEMP. EXEC SQL CLOSE THISEMP END-EXEC.	"Step 6: Close the Cursor" on page 3-25.

Step 1: Define the Cursor

To define and identify a set of rows to be accessed with a cursor, issue a DECLARE CURSOR statement:

1. The DECLARE CURSOR statement names a cursor and specifies a SELECT statement. The SELECT statement defines a set of rows that, conceptually, makes up the results table. The statement looks like this:

```
EXEC SQL
  DECLARE cursor-name CURSOR FOR
    SELECT column-1, column-2
      FROM table-name
      WHERE column-1 = search-condition
    FOR UPDATE OF column-2
END-EXEC.
```

   The SELECT statement shown here is rather simple. However, you can code several other types of clauses in a SELECT statement within a DECLARE CURSOR statement.

2. If you intend to update any columns in any or all of the rows of the identified table (the table named in the FROM clause), include the FOR UPDATE OF clause; it names each column you intend to update. If you do not specify the names of columns you will later update, and you specify the ORDER BY clause, a negative SQLCODE is returned if an update is attempted. If you do not specify the FOR UPDATE OF clause and you do not specify the ORDER BY clause, you can update any of the columns of the specified table.

   You can update a column of the identified table even though it is not part of the results table. In this case, you do not need to name the column in the SELECT statement. When the cursor retrieves a row (using FETCH) that contains a column value you want to update, you can use UPDATE ... WHERE CURRENT OF to update the row.

   For example, assume that each row of the results table includes the *EMPNO*, *LASTNAME*, and *DEPTNO* columns from the USER1.TEMPL table. If you want to update the *JOBCODE* column (one of the columns in each row of the USER1.TEMPL table), the DECLARE CURSOR statement should include FOR UPDATE OF JOBCODE ... even though JOBCODE is omitted from the SELECT statement.

3. The results table is read-only if the SELECT statement includes the keyword DISTINCT, a UNION operator, a column function, a GROUP BY clause, or a HAVING clause. The results table is also read-only if the FROM clause of the SELECT statement identifies a read-only view or identifies more than one table or view (that is, if it is "joined" with another table). For details about the join technique, see "Joining Data from More Than One Table" on page 3-29.

Step 2: Open the Cursor

To tell SQL that you are ready to process the first row of the results table, issue the OPEN statement. When your program issues the OPEN statement, SQL processes the SELECT statement within the DECLARE CURSOR statement to identify a set of rows, called a results table[3], using the current value of any host variables specified in the SELECT statement. The OPEN statement looks like this:

```
EXEC SQL
  OPEN cursor-name
END-EXEC.
```

Step 3: Specify What to Do when End-of-Data Is Reached

To find out when no rows are left to process, test the SQLCODE field for a value of 100 (that is, end-of-data). This condition occurs when the FETCH statement has retrieved the last row in the results table and your program issues a subsequent FETCH. For example:

```
...
IF SQLCODE =100 GO TO DATA-NOT-FOUND.
```

An alternative to this technique is to code the WHENEVER NOT FOUND clause. The WHENEVER NOT FOUND clause can result in a branch to another part of your program, where a CLOSE statement is issued. The WHENEVER NOT FOUND clause looks like this:

```
EXEC SQL
  WHENEVER NOT FOUND GO TO symbolic-address
END-EXEC.
```

Your program should anticipate an end-of-data condition whenever a cursor is used to fetch a row, and should be prepared to handle this situation when it occurs.

Step 4: Retrieve a Row Using the Cursor

To move the contents of a selected row into your program's host variables, use the FETCH statement. The SELECT statement within the DECLARE CURSOR statement identifies rows that contain the column values your program wants (that is, the results table is defined), but SQL does not retrieve any data for your application program until FETCH is issued.

When your program issues the FETCH statement, SQL uses the cursor to point to the next row in the results table, making it the **current row**. SQL then moves the current row's contents into your program's host variables (specified with the INTO clause). This sequence is repeated each time FETCH is issued, until you have processed all rows in the results table.

SQL maintains the position of the current row (that is, the cursor points to the current row) until the next FETCH statement for the cursor is issued. The UPDATE statement does not change the position of the current row within the results table, although the DELETE statement does.

---

[3] A result table can contain zero, one, or many rows, depending on the extent to which the search condition is satisfied.

The FETCH statement looks like this:

```
EXEC SQL
  FETCH cursor-name
    INTO :host variable-1, :host variable-2
END-EXEC.
```

Step 5a: Update the Current Row

When your program has retrieved the current row, you can update its data by using the UPDATE statement with the WHERE CURRENT OF clause. The WHERE CURRENT OF clause specifies a cursor that points to the row you want to update. The UPDATE ... WHERE CURRENT OF statement looks like this:

```
EXEC SQL
  UPDATE table-name
    SET column-1 = value [, column-2 = value] ...
    WHERE CURRENT OF cursor-name
END-EXEC.
```

When used with a cursor, the UPDATE statement:

- Updates only one row—the current row
- Identifies a cursor that points to the row to be updated
- Requires that the column updated be named previously in the FOR UPDATE OF clause of the DECLARE CURSOR statement, if an ORDER BY clause was also specified

After you have updated a row, the cursor's position remains on that row (that is, it points to the current row) until you issue a FETCH statement for the next row.

Step 5b: Delete the Current Row

When your program has retrieved the current row, you can delete the row by using the DELETE statement. To do this, you issue a DELETE statement designed for use with a cursor; the WHERE CURRENT OF clause specifies a cursor that points to the row you want to delete. The DELETE ... WHERE CURRENT OF statement looks like this:

```
EXEC SQL
  DELETE FROM table-name
    WHERE CURRENT OF cursor-name
END-EXEC.
```

When used with a cursor, the DELETE statement:

- Deletes only one row—the current row.
- Uses the WHERE clause to identify a cursor that points to the row to be deleted.

After you have deleted a row, you cannot update or delete another row using that cursor until you issue a FETCH statement for the next row.

"The DELETE Statement" on page 3-4 showed you how to use the DELETE statement to delete all rows that meet a specific search condition. Alternatively, you can use the FETCH and DELETE ... WHERE CURRENT OF statements when you want to obtain a copy of the row, examine it, then delete it.

Step 6: Close the Cursor

If you are finished processing the rows of the results table and you want to use the cursor again, issue a CLOSE statement to close the cursor:

```
EXEC SQL
  CLOSE cursor-name
END-EXEC.
```

If you are finished processing the rows of the results table and you do not want to use the cursor again, you can let the system automatically close the cursor when the first SQL program in the program stack ends.

Because an open cursor still holds locks on referred to tables or views, you should explicitly close any open cursors as soon as they are no longer needed.

A Unit of Recovery and Open Cursors

If your program completes a unit of recovery (that is, it either commits or rolls back the changes made so far), all open cursors are automatically closed by SQL, unless you have specified HOLD. You can reopen the cursor, but you will begin processing at the beginning of the results table.

If you wish to continue processing from the current cursor position after a COMMIT or ROLLBACK, you must specify COMMIT HOLD or ROLLBACK HOLD. When HOLD is specified, any open cursors are left open and keep their cursor position so that processing can resume. All record locks are still released. Since a FETCH statement acquires a lock on the row that it retrieves, you should not normally issue a FETCH statement before a COMMIT.

Preventing Duplicate Rows

When SQL evaluates a SELECT statement, several rows might qualify to be in the results table, depending on the number of rows that satisfy the SELECT statement's search condition. Some of the rows in the results table might be duplicates. You can specify that you do not want any duplicates by using the DISTINCT keyword, followed by the list of column names:

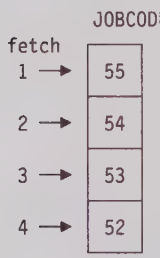
```
SELECT DISTINCT JOBCODE, SEX
 ...
```

DISTINCT means you want to select only unique rows. If a selected row duplicates another row in the results table, the duplicate row is ignored (it is not put into the results table). For example, suppose you want a list of employee job codes. You do not need to know which employee has what job code. Because it is probable that several people in a department have the same job code, you can use DISTINCT to ensure that the results table has only unique values.

The following example shows how to do this:

```
EXEC SQL
  DECLARE XMP2 CURSOR FOR
    SELECT DISTINCT JOBCODE
      FROM USER1.TEMPL
      WHERE DEPTNO = :JOB-DEPT
END-EXEC.
 ...
EXEC SQL
  FETCH XMP2
    INTO :JOBCODE
END-EXEC.
```

The result is four rows (in this example, JOB-DEPT is set to D11).



If you do not include DISTINCT in a SELECT clause, you might find duplicate rows in your result, because SQL retrieves the *JOBCODE* column's value for each row that satisfies the search condition.

Performing Complex Search Conditions

The following section explains more advanced things you can do with search conditions.

Keywords for Use in Search Conditions

A search condition can contain any of the keywords BETWEEN...AND, IN, and LIKE.

Note: Literal values are shown in the following examples to keep the examples simple. However, you could just as easily code host variables instead. Remember to precede each host variable with a colon.

- **BETWEEN ... AND ...** is used to specify a search condition that is satisfied by any value that falls on or between two other values. For example, to find all employees who were hired in 1987, you could use this:

  ```
  ...
  WHERE HIREDATE BETWEEN '870101' AND '871231'
  ```

  The BETWEEN keyword is inclusive. A more cumbersome, but explicit, search condition that produces the same result is:

  ```
  ...
  WHERE HIREDATE >= '870101' AND HIREDATE <= '871231'
  ```

- **IN** says you are interested in rows in which the value of the specified expression is among the values you have listed. For example, to find the names of all employees in departments A00, C01, and E21, you could use:

```
...
 WHERE DEPTNO IN ('A00', 'C01', 'E21')
```

- **LIKE** says you are interested in rows in which a column value is similar to the value you supply. When you use LIKE, the value on the left is always a column name of character type. The value on the right must be a character value: either a character string, a mixed character string, a host variable that contains character data, or the USER special register.

  When you use LIKE, SQL searches for a character string similar to the one you specify. The degree of similarity is determined by two special characters used in the string that you include in the search condition:

  _ An underscore character stands for any single character.

  % A percent sign stands for an unknown string of 0 or more characters. SQL expects the unknown string not to begin in the first position in the column unless the % precedes the string. In that case, the unknown string can begin anywhere in the column.

  Note: If you are operating on mixed data, the following distinction applies: An EBCDIC underscore character refers to one EBCDIC character. No such restriction applies to the percent sign; that is, a percent sign refers to any number of characters of any type.

  Use the underscore character or percent sign either when you do not know or do not care about all the characters of the column's value. For example, to find out which employees live in Minneapolis, you could use:

```
...
 WHERE ADDRESS LIKE '%MINNEAPOLIS%'
```

  In this case, you should be sure that MINNEAPOLIS was not part of a street address or part of another city name. SQL returns any row with the string MINNEAPOLIS in the ADDRESS column, no matter where the string occurs.

  In another example, to list the towns whose names begin with SAN_, you could use:

```
...
 WHERE TOWN LIKE 'SAN %'
```

Multiple Search Condition within a WHERE Clause

You have seen how to qualify a request using one search condition. You can qualify your request further by coding a search condition that includes several predicates. The search condition you specify can contain any of the comparison operators or the keywords BETWEEN, IN, and LIKE.

You can join any two predicates with the connectors AND and OR. In addition, you can use the NOT keyword to specify that the desired search condition is the negated value of the specified search condition. A WHERE clause can have as many predicates as you want.

- **AND** says that, for a row to qualify, the row must satisfy both predicates of the search condition. For example, to find out which employees in department D21 were hired after December 31, 1987, you would say:

```
...
WHERE DEPTNO = 'D21' AND HIREDATE > '871231'
```

- **OR** says that, for a row to qualify, the row can satisfy the condition set by either or both predicates of the search condition. For example, to find out which employees are in either department C01 or D11, you could code[4]:

```
...
WHERE DEPTNO = 'C01' OR DEPTNO = 'D11'
```

- **NOT** says that, to qualify, a row must not meet the criteria set by the search condition or predicate that follows the NOT. For example, to find all employees in department E11 except those with a job code lower than 55, you could say:

```
...
WHERE DEPTNO = 'E11' AND NOT JOBCODE < 55
```

When SQL evaluates search conditions that contain these connectors, it does so in a specific order. SQL first evaluates the NOT clauses, next evaluates the AND clauses, and then the OR clauses.

You can change the order of evaluation by using parentheses. The search condition(s) enclosed in parentheses are evaluated first. For example, to select all employees in departments E11 and E21 who have job codes greater than 53, you could say:

```
...
WHERE JOBCODE > 53 AND
  (DEPTNO = 'E11' OR DEPTNO = 'E21')
```

The parentheses determine the meaning of the search condition. In this example, you want all rows that have a:

DEPTNO value of E11 or E21, and
JOBCODE value greater than 53

If you did not use parentheses:

```
...
WHERE JOBCODE > 53 AND DEPTNO = 'E11'
  OR DEPTNO = 'E21'
```

your result is different. The selected rows are rows that have:

DEPTNO = E11 and JOBCODE > 53, or
DEPTNO = E21, regardless of the JOBCODE value

---

[4] You could also use IN to specify this request: WHERE DEPTNO IN ('C01', 'D11').

Joining Data from More Than One Table

In the SELECT examples you have seen so far, the retrieved information has usually been in one table. Sometimes the information you want is not in one table only. To form a row of the results table, you might want to retrieve some column values from a row in one table and some columns from a row in another table. You can use a SELECT statement to retrieve and join column values from two or more tables into a row.

Joining data (as described in this section) differs from using the UNION keyword in the following ways:

With UNION keyword, the column values specified in one SELECT statement must have compatible data types with the column values in the other SELECT statement. Also, the set of rows selected from one table is appended to the set selected from the other table, and duplicate rows are eliminated.

With join, column values from one row of a table are combined with column values from another row of another (or the same) table to form a single row of data. SQL examines both tables specified in the join SELECT statement to retrieve data from as many rows as meet the search criteria specified in the WHERE clause.

For example, suppose you want to retrieve, for project MA2112, the employee numbers, names, activity codes, and amount of time spent on that project. In other words, you want the *EMPNO* and *LASTNAME* columns from the USER1.TEMPL table and the *ACTNO* and *EMPTIME* columns from the USER1.TEMPRACT table. To find this information, you need to join the two tables.

To do this, you list the two tables you are joining in the FROM clause of the SELECT statement. If the tables have any column names that are the same, you qualify those column names by prefixing them with the name of the table:

```
EXEC SQL
  DECLARE XMP7 CURSOR FOR
    SELECT USER1.TEMPL.EMPNO, LASTNAME, ACTNO, EMPTIME
      FROM USER1.TEMPL, USER1.TEMPRACT
      WHERE USER1.TEMPL.EMPNO = USER1.TEMPRACT.EMPNO
        AND USER1.TEMPRACT.PROJNO = 'MA2112'
END-EXEC.
 ...
EXEC SQL
  FETCH XMP7
    INTO :PERSON, :NAME3, :ACTIV, :TIME
END-EXEC.
```

To better understand what results from these SQL statements, imagine that SQL goes through the following process:

Step 1: SQL selects all rows in USER1.TEMPRACT with a PROJNO value of MA2112:

```
...
SELECT USER1.TEMPL.EMPNO, ...
    ACTNO, EMPTIME
  FROM ... USER1.TEMPRACT
  WHERE ...
   USER1.TEMPRACT.PROJNO = 'MA2112'
```

Which results in an interim results table:

(from USER1.TEMPRACT)

EMPNO	ACTNO	EMPTIME
000170	70	100
000190	70	100

Step 2: SQL selects all rows from the USER1.TEMPL table with a EMPNO value equal to the EMPNO values from step 1.

From each row, SQL selects the row's EMPNO and LASTNAME values.

```
...
SELECT USER1.TEMPL.EMPNO, LASTNAME
   FROM USER1.TEMPL, ...
   WHERE USER1.TEMPL.EMPNO =
    USER1.TEMPRACT.EMPNO ...
...
```

Which results in another interim results table:

(from USER1.TEMPL)

EMPNO	LASTNAME
000170	YOSHIMURA
000190	WALKER

Step 3: SQL combines the results of the previous two steps to form a results table that can be retrieved a row at a time with a FETCH statement:

```
EXEC SQL
  DECLARE XMP7 CURSOR FOR
    SELECT USER1.TEMPL.EMPNO, LASTNAME, ACTNO, EMPTIME
      FROM USER1.TEMPL, USER1.TEMPRACT
        WHERE USER1.TEMPL.EMPNO = USER1.TEMPRACT.EMPNO
        AND USER1.TEMPRACT.PROJNO = 'MA2112'
END-EXEC.
...
EXEC SQL
  FETCH XMP7
    INTO :PERSON, :NAME3, :ACTIV, :TIME
END-EXEC.
```

The combined results table:

	PERSON	NAME3	ACTIV	TIME
fetch 1→	000170	YOSHIMURA	70	100
2→	000190	WALKER	70	100

Join and the WHERE Clause

The WHERE clause establishes a condition for joining two or more tables; that is, it gives a relationship between the two tables.

When tables are joined, the WHERE clause can be used to:

- Concatenate, on a one-to-one basis, a row from table A with a corresponding row from table B
- Concatenate distinct groups of rows from table A to different rows of table B
- Concatenate groups of rows from table A to selected rows of table B

In the preceding example, the statement tells SQL that you want columns from rows in the USER1.TEMPL table concatenated with columns from rows in the USER1.TEMPRACT table that have the same EMPNO value and a PROJNO value of MA2112.

If you do not include a WHERE clause, *each row* of the USER1.TEMPL table is concatenated with *every row* of the USER1.TEMPRACT table to form a results table, even though the data may be unrelated. In other words, if you do not include a WHERE clause, the number of rows in the results table is the product of the number of rows of each joined table.

For example, note that the lack of a WHERE clause in the following SQL statement causes a concatenation of each row of table A to every row of table B:

```
EXEC SQL
 SELECT * FROM A, B
END-EXEC.
```

The results look like this:

Table A consists of:

Column 1	Column 2
A1	AA1
A2	AA2
A3	AA3

Table B consists of:

Column 1	Column 2
B1	BB1
B2	BB2

The results of the SELECT are:

VAR1	VAR2	VAR3	VAR4
A1	AA1	B1	BB1
A1	AA1	B2	BB2
A2	AA2	B1	BB1
A2	AA2	B2	BB2
A3	AA3	B1	BB1
A3	AA3	B2	BB2

Some of the uses you may have for joining two or more tables are:

- To create a view

  Note: The view created in this manner cannot be processed using UPDATE, DELETE, or INSERT statements.

- Merely to select data
- To join a table or view to itself

Notes on the Join Technique

When you join two or more tables:

- If there are common column names, you must prefix each common name with the name of the table (or a correlation name). Column names that are unique do not need a prefix.
- If you do not list the column names you want, but instead use SELECT *, SQL returns rows that consist of all the columns of the first table, followed by all the columns of the second table, and so on.
- If the GROUP BY clause is used in a view's definition, you cannot join the view with any other table.
- You must be authorized to select rows from each table or view specified in the FROM clause.

Inserting Multiple Rows into a Table

You can use a SELECT statement within an INSERT statement to insert zero, one, or more rows selected from the table or view you specify into another table. The rows you select cannot be from the same table you are inserting into. SQL will not process the INSERT statement if the tables are the same.

One use for this kind of INSERT statement is to move data into a table you have created for summary data. For example, suppose you want a table that shows each employee's time commitments to projects. You could create a table called EMPTIME with the columns *EMPNUMBER*, *PROJNUMBER*, *STARTDATE*, *ENDDATE*, and *TTIME*, and then use the following INSERT statement to fill the table:

```
EXEC SQL
  INSERT INTO USER1.EMPTIME
      (EMPNUMBER, PROJNUMBER, STARTDATE, ENDDATE)
    SELECT EMPNO, PROJNO, EMSTDATE, EMENDATE
      FROM USER1.TEMPRACT
END-EXEC.
```

The SELECT statement embedded in the INSERT statement is no different from the SELECT statement you use to retrieve data. With the exception of UNION and ORDER BY, you can use all the keywords, column functions, and techniques used to retrieve data. SQL inserts all the rows that meet the search conditions into the table you specify. Inserting rows from one table into another does not affect any existing rows in either the source table or the target table.

Notes on Multiple Row Insertion

You should keep in mind the following notes when inserting multiple rows into a table:

- The number of columns implicitly or explicitly listed in the INSERT statement must equal the number of columns listed in the SELECT statement.
- The data in the columns you are selecting must be compatible with the columns you are inserting into.
- In the event the SELECT statement embedded in the INSERT returns no rows, an SQLCODE of 100 is returned to alert you that no rows were inserted. If you successfully insert rows, the SQLERRD(3) field of the SQLCA will have an integer representing the number of rows SQL actually inserted.
- If SQL finds an error while running the INSERT statement, SQL stops the operation. If you specify COMMIT (*CHG) or COMMIT (*ALL), nothing is inserted into the table and a negative SQLCODE is returned. If you specify COMMIT(*NONE), any rows inserted prior to the error remain in the table.
- You can join two or more tables with a SELECT statement in an INSERT statement. Loaded in this manner, the table can be operated on with UPDATE, DELETE, and INSERT statements, because the rows exist as physically stored rows in a table.
- When inserting new rows, you cannot specify an insertion sequence.

Inserting Default Values into Columns

At the time a table is created, individual columns within that table can be defined such that default values are inserted if no other values are specified on an INSERT statement.

To cause SQL to insert a default value into a column, omit that column's name and value from your INSERT statement. Lacking any explicit assignment, SQL will automatically insert the default value into that column.

Note: If a table column was defined as NOT NULL rather than NOT NULL WITH DEFAULT, you must explicitly provide a value for this column in any rows you INSERT. If you omit such specifications, SQL returns a negative SQLCODE.

Also note that, when you insert a row into a view, there is an additional consideration. If the view does not contain all the columns of the base table, SQL inserts default values into those columns of the base table that are not in the view.

Chapter 4. Common Concepts and Rules for Using SQL with Host Languages

This chapter describes some concepts and rules that are common to using SQL statements in a host language that involve:

- Using host variables in SQL statements
- Handling SQL error and return codes
- Handling exception conditions with the WHENEVER statement

Using Host Variables in SQL Statements

When your program retrieves data, the values are put into data items defined by your program and specified with the INTO clause of a SELECT or FETCH statement. The data items are called **host variables**.

A host variable is a field in your program that is referenced in an SQL statement, usually as the source or target for the value of a column. The host variable and column must have the same data type.

A **host structure** is a group of data items used as the source or target for a set of selected values (for example, the set of values for the columns of a row).

Note: By using a host variable instead of a literal value in an SQL statement, you give the application program the flexibility it needs to process different rows in a table or view.

For example, instead of coding an actual department number in a WHERE clause, you can use a host variable set to the department number you are currently interested in.

Host variables are commonly used in SQL statements in these ways:

1. **In a WHERE clause:** You can use a host variable to specify a value in the predicate of a search condition, or to replace a literal value in an expression. For example, if you have defined a field called EMPID that contains an employee number, you can retrieve the name of the employee whose number is 000110 with:

```
MOVE '000110' TO EMPID.
EXEC SQL
  SELECT LASTNAME
    INTO :PGM-LASTNAME
    FROM USER1.TEMPL
    WHERE EMPNO = :EMPID
END-EXEC.
```

2. **As a receiving area for column values (named in an INTO clause):** You can use a host variable to specify a program data area that is to contain the column values of a retrieved row. The INTO clause names one or more host variables that you want to contain column values returned by SQL. For example, suppose you are retrieving the *EMPNO*, *LASTNAME*, and *DEPTNO* column values from rows in the USER1.TEMPL table. You could define a data area in your program to

hold each column, then name the data areas with an INTO clause. For example:

```
EXEC SQL
  SELECT EMPNO, LASTNAME, DEPTNO
    INTO :CBLEMPNO, :CBLNAME, :CBLDEPT
    FROM USER1.TEMPL
    WHERE EMPNO = :EMPID
END-EXEC.
```

In this example, the host variable CBLEMPNO receives the value from EMPNO, CBLNAME receives the value from LASTNAME, and CBLDEPT receives the value from DEPTNO.

3. **As a value in a SELECT clause:** When specifying a list of items in the SELECT clause, you are not restricted to the column names of tables and views. Your program can return a set of column values intermixed with host variable values and literal constants. For example:

```
MOVE '000220' TO PERSON.
EXEC SQL
  SELECT 'A', LASTNAME, SALARY, :RAISE,
      SALARY + :RAISE
    INTO :PROCESS, :PERSON-NAME, :EMP-SAL,
      :EMP-RAISE, :EMP-TTL
    FROM USER1.TEMPL
    WHERE EMPNO = :PERSON
END-EXEC.
```

The results are:

PROCESS	PERSON-NAME	EMP-SAL	EMP-RAISE	EMP-TTL
A	LUTZ	29840	4476	34316

4. **As a value in other clauses of an SQL statement:**

   The SET clause in an UPDATE statement
   The VALUES clause in an INSERT statement

   For more information on clauses, see the *SQL/400 Reference* manual.

Assignment Rules

SQL column values are sent to (or assigned to) host variables during the running of FETCH and SELECT statements. SQL column values are set from (or assigned from) host variables during the running of INSERT and UPDATE statements. All assignment operations observe the following rules:

- Numbers and strings are not compatible:

  Numbers cannot be assigned to string columns or string host variables.
  Strings cannot be assigned to numeric columns or numeric host variables.

- All character strings are compatible; all numeric values are compatible. Conversions are performed by SQL whenever necessary.

- A null value cannot be assigned to a host variable that does not have an associated indicator variable.

Rules for Character String Assignment

Rules regarding character string assignment are:

- When a character string is put into a column, the length of the character string value must not be greater than the length attribute of the column.
- When a mixed character results column is assigned to a mixed column, the value of the mixed character results column must be a valid mixed character string.
- When the value of a results column is put into a host variable and the character string value of the results column is longer than the length attribute of the host variable, the string is truncated on the right by the necessary number of characters. If this occurs, SQLWARN0 and SQLWARN1 (in the SQLCA) are set to W.
- When the value of a results column is put into a fixed-length host variable or when the value of a host variable is put into a fixed-length CHAR results column and the length of the character string value is less than the length attribute of the target, the character string is padded on the right with the necessary number of blanks.
- When a mixed character results column is truncated because the length of the host variable into which it was being inserted was less than the length of the string, the shift-in character at the end of the string is preserved. The result is, therefore, still a valid mixed character string.

Rules for Numeric Assignment

Rules regarding numeric assignment are:

- **The whole part of a number may be altered when converting it to single-precision floating-point.** Because a single-precision floating-point field can only contain seven decimal digits, any whole part of a number that contains more than seven digits is altered due to rounding.
- **The whole part of a number is never truncated.** If necessary, the fractional part of a number is truncated. If the number, as converted, does not fit into the target host variable or column, a negative SQLCODE is returned.
- Whenever a **decimal, numeric, or binary number** is assigned to a decimal, numeric, or binary column or host variable, the number is converted, if necessary, to the precision and scale of the target. The necessary number of leading zeros is appended or deleted; in the fractional part of the number, the necessary number of trailing zeros is appended, or the necessary number of trailing digits is eliminated.
- When a **binary or floating-point number** is assigned to a decimal or numeric column or host variable, the number is first converted to a temporary decimal or numeric number and then converted, if necessary, to the precision and scale of the target.
  - When a **halfword binary integer** (SMALLINT) with 0 scale is converted to decimal or numeric, the temporary result has a precision of 5 and a scale of 0.
  - When a **fullword binary integer** (INTEGER) is converted to decimal or numeric, the temporary result has a precision of 11 and a scale of 0.
  - When a **floating-point number** is converted to decimal or numeric, the temporary result has a precision of 31 and the maximum scale that allows the whole part of the number to be represented without loss of either significance or accuracy.

Indicator Variables and Structures

An **indicator variable** is a halfword integer variable used to indicate whether its associated host variable has been assigned a null value:

- If a host variable is used any place other than in an INTO clause, it must not have an associated indicator variable containing a negative value. You are responsible for ensuring that this indicator is not set to a negative value.
- If the value for the results column is null, SQL puts a negative value in the indicator variable.
- If you do not use an indicator variable and the results column is a null value, a negative SQLCODE is returned.
- If the value for the results column causes a numeric conversion error or an arithmetic expression error, SQL sets the indicator variable to −2.

You can also use an indicator variable to verify that a retrieved character string value has not been truncated. If the indicator variable contains a positive integer, it specifies the original length of the string.

When SQL returns a value from a results column, you can test the indicator variable. If the value of the indicator variable is less than zero, you know the value of the results column is null. When SQL returns a null value, nothing is put into the host variable used to contain the value of the results column and the value of the host variable is unchanged.

You specify an indicator variable (preceded by a colon) immediately after the host variable. For example:

```
EXEC SQL
  SELECT COUNT(*), AVG(SALARY)
    INTO :PLICNT, :PLISAL:INDNULL
    FROM USER1.TEMPL
    WHERE JOBCODE < 52
END-EXEC.
```

You can then test INDNULL to see if it contains a negative value. If it does, you know SQL returned a null value.

Used with Host Structures

You can also specify an **indicator structure** (defined as an array of halfword integer variables) to support a host structure. If the results column values returned to a host structure can be null, you can append an indicator structure name to the host structure name. This allows SQL to notify your program about each null value returned to a host variable in the host structure.

For example, in COBOL:

```
01 PEMPL.
    10 EMPNO              PIC X(6).
    10 FIRSTNME.
       49 FIRSTNME-LEN    PIC S9(4) USAGE COMP-4.
       49 FIRSTNME-TEXT   PIC X(12).
    10 MIDINIT            PIC X(1).
    10 LASTNAME.
       49 LASTNAME-LEN    PIC S9(4) USAGE COMP-4.
       49 LASTNAME-TEXT   PIC X(15).
    10 DEPTNO             PIC X(3).
    77 EMPIND             PIC S9999 USAGE COMP-4 OCCURS 5 TIMES.
  ...
  MOVE '000220' TO EMPNO.
  ...
  EXEC SQL
    SELECT *
      INTO :PEMPL:EMPIND
      FROM USER1.TEMPL
      WHERE EMPNO = :EMPNO
  END-EXEC.
```

In this example, EMPIND is an array containing 5 values, each of which can be tested for a negative value. If, for example, EMPIND(5) contains a negative value, the corresponding host variable in the host structure (that is, DEPTNO) contains a null value for the selected row.

In the above example, SQL selects the column values of the row into a host structure. Therefore, you must use a corresponding structure for the indicator variables to determine which (if any) selected column values are null.

Handling SQL Error Return Codes

When an SQL statement is processed in your program, SQL places a return code in the SQLCODE field of the SQL communication area (SQLCA). The SQLCA allows SQL to communicate with your program. The return code indicates the success or failure of the running of your statement. If SQL encounters an error while processing the statement, the return code in SQLCODE is a negative number. If SQL encounters an exception but valid condition while processing your statement, the return code is a positive number. If your SQL statement is processed without encountering an error or exception condition, the return code is zero.

Note: There are situations when a zero return code is returned to your program and the result might not be satisfactory. For example, if a value was truncated as a result of running your program, the SQLCODE returned to your program is zero. However, one of the SQL warning flags (SQLWARN1) indicates truncation.

Warning: If you do not test for negative SQL return codes or specify a WHENEVER SQLERROR statement, your program will continue to the next statement. Continuing to run after an error can produce unpredictable results.

Because the SQLCA is a valuable problem-diagnosis tool, it is a good idea to include in your application programs the instructions necessary to display some of the information contained in the SQLCA. Especially important are the following SQLCA fields:

SQLCODE	Return code.
SQLERRD(3)	The number of rows updated, inserted, or deleted by SQL.
SQLWARN0	If set to W, at least one of the SQL warning flags (SQLWARN1 through SQLWARN7) is set.

For more information about SQLCA, see Appendix B. SQLCA and SQLDA Control Blocks in the *SQL/400 Reference* manual.

Handling Exception Conditions with the WHENEVER Statement

The WHENEVER statement causes SQL to check the SQLCA and continue processing your program, or branch to another area in your program if an error, exception, or warning exists as a result of running an SQL statement. An exception condition handling subroutine (part of your program) can then examine the SQLCODE field to take an action specific to the error or exception situation.

The WHENEVER statement allows you to specify what you want to do whenever a general condition is true. You can specify more than one WHENEVER statement for the same condition. When you do this, the first WHENEVER statement applies to all subsequent SQL statements in the source program until another WHENEVER statement is specified.

The WHENEVER statement looks like this:

```
EXEC SQL
  WHENEVER condition action
END-EXEC.
```

There are three conditions you can specify:

SQLWARNING — Specify SQLWARNING to indicate what you want done when SQLWARN0 = W or SQLCODE contains a positive value other than 100.

Note: SQLWARN0 could be set for several different reasons. For example, if the value of a column was truncated when it was moved into a host variable, your program might not regard this as an error.

SQLERROR — Specify SQLERROR to indicate what you want done when an error code is returned as the result of an SQL statement (SQLCODE < 0).

NOT FOUND — Specify NOT FOUND to indicate what you want done when a SQLCODE of +100 is returned because:

- After a single-row SELECT is issued or after the first FETCH is issued for a cursor, the data the program specifies does not exist.
- After a subsequent FETCH, no more rows satisfying the cursor SELECT statement are left to retrieve.
- After an UPDATE, a DELETE, or an INSERT, no row meets the search condition.

You can also specify the action you want taken:

CONTINUE — This causes your program to continue to the next statement.

GO TO label — This causes your program to branch to an area in the program. The label for that area may be preceded with a colon. The WHENEVER ... GO TO statement:

- In COBOL, must be a section name or an unqualified paragraph name
- In PL/I, is a label
- In RPG, is the label of a TAG

For example, if you are retrieving rows using a cursor, you expect that SQL will eventually be unable to find another row when the FETCH statement is issued. To prepare for this situation, specify a WHENEVER NOT FOUND GO TO ... statement to cause SQL to branch to a place in the program where you issue a CLOSE statement in order to close the cursor properly.

Note: A WHENEVER statement affects all subsequent *source* SQL statements until another WHENEVER is encountered.

In other words, all SQL statements coded between two WHENEVER statements (or following the first, if there is only one) are governed by the first WHENEVER statement, regardless of the path the program takes.

Because of this, the WHENEVER statement *must precede* the first SQL statement it is to affect. If the WHENEVER *follows* the SQL statement, the branch is not taken on the basis of the value of the SQLCODE set by that SQL statement. However, if your program checks the SQLCODE directly, the check must be done after the SQL statement is run.

The WHENEVER statement does not provide a CALL to a subroutine option. For this reason, you might want to examine the SQLCODE value after each SQL statement is run and call a subroutine, rather than use a WHENEVER statement.

Chapter 5. Using SQL Statements in COBOL Programs

The AS/400 system supports more than one COBOL. SQL/400 only supports the COBOL/400 language. This chapter describes the unique application and coding requirements for embedding SQL statements in a COBOL/400 program. Requirements for host structures and host variables are defined.

A detailed sample COBOL program, showing how SQL statements can be used, is provided in Appendix D.

Application Requirements

To run SQL statements, your COBOL program must have an SQL communication area (SQLCA). The SQLCA is declared in the WORKING STORAGE section in the program's DATA DIVISION.

SQL Communication Area (SQLCA)

The SQLCA is where SQL returns information about the results of running each SQL statement. To include the SQLCA declaration, you can use the INCLUDE statement:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
 ...
     EXEC SQL
     INCLUDE SQLCA
     END-EXEC.
```

You can specify INCLUDE SQLCA wherever a 77 level or a record description entry can be specified in the WORKING STORAGE section. (Other INCLUDE statements can be specified anywhere in the DATA DIVISION or the PROCEDURE DIVISION.)

When you use the INCLUDE statement, the SQL COBOL precompiler includes COBOL source statements for the SQLCA:

```
01 SQLCA.
   05 SQLCAID        PIC X(8).
   05 SQLCABC        PIC S9(9) COMP-4.
   05 SQLCODE        PIC S9(9) COMP-4.
   05 SQLERRM.
      49 SQLERRML    PIC S9(4) COMP-4.
      49 SQLERRMC    PIC X(70).
   05 SQLERRP        PIC X(8).
   05 SQLERRD        OCCURS 6 TIMES
                     PIC S9(9) COMP-4.
   05 SQLWARN.
      10 SQLWARN0    PIC X(1).
      10 SQLWARN1    PIC X(1).
      10 SQLWARN2    PIC X(1).
      10 SQLWARN3    PIC X(1).
      10 SQLWARN4    PIC X(1).
      10 SQLWARN5    PIC X(1).
      10 SQLWARN6    PIC X(1).
      10 SQLWARN7    PIC X(1).
   05 SQLEXT         PIC X(8).
```

For more information on SQLCA, see Appendix B. SQLCA and SQLDA Control Blocks in the *SQL/400 Reference* manual.

Coding Requirements

You must put SQL statements (other than the INCLUDE statement) in the PROCEDURE DIVISION. The SQL statement can be preceded by a paragraph name if a COBOL statement in the same place could.

When you issue SQL statements from a COBOL program, begin each statement with EXEC SQL.

Each SQL statement must have an ending delimiter. In COBOL programs, end each SQL statement with END-EXEC. If several SQL statements are coded with no COBOL statements between them, the ending delimiter of each SQL statement must include an ending period (END-EXEC.). For a single SQL statement between two COBOL statements, an ending period is optional (END-EXEC. or END-EXEC). The ending period might not be appropriate in a structured program or where an SQL statement is embedded in a COBOL clause. An ending period ends the preceding COBOL statement (an IF ... THEN statement, for example).

For example, an UPDATE statement issued from a COBOL program would look like this:

```
EXEC SQL
UPDATE USER1.TDEPT
  SET MGRNO = :MGR-NUM
  WHERE DEPTNO = :INT-DEPT
END-EXEC.
```

Including Code: You cannot use COBOL COPY verbs to include SQL statements. You must use the SQL INCLUDE statement. INCLUDE statements cannot be nested.

Continuation for SQL Statements: The line continuation rules for SQL statements are the same as those for other COBOL statements, except that EXEC SQL must be specified within one line. If you continue a string constant from one line to the next, the first nonblank character in the next line must be a string delimiter. If you continue a delimited identifier from one line to the next, the first nonblank character in the next line must be the SQL escape character.

Comments: You can include COBOL comment lines (* in column 7) within SQL statements.

Note: COBOL debugging lines (D in column 7) and page eject lines (/ in column 7) are treated as comment lines by the SQL precompiler.

Margins: Code SQL statements in columns 12 through 72. If EXEC SQL starts before the specified margin (that is, before column 12), the SQL precompiler will not recognize the statement.

Sequence Numbers: The source statements generated by the SQL precompiler are generated with the same sequence number as the SQL statement.

Reserved Words: Do not use host variable names that begin with SQL. In addition, a list of SQL reserved words can be found under Appendix A.

Dynamic SQL in a COBOL Program: Dynamic SQL is an advanced programming technique described in Chapter 8. With dynamic SQL, your program can develop and then run SQL statements while the program is running. A SELECT statement with a variable SELECT list (that is, a list of the data to be returned as part of the query) that runs dynamically requires an SQL descriptor area (SQLDA), because you cannot know in advance how many or what type of variables to allocate in order to receive the results of the SELECT. Because the SQLDA uses pointer variables, which are not supported by COBOL, an INCLUDE SQLDA statement cannot be specified in a COBOL program. An SQLDA must be set up by a PL/I program and passed to the COBOL program in order to use it.

Using COBOL Host Variables in SQL Statements: When your program retrieves data, the values are put into data items defined by your program and specified with the INTO clause of a SELECT or FETCH statement. The data items are called **host variables**. See "Using Host Variables in SQL Statements" on page 4-1 for more information on using host variables.

Host Structures

A **host structure** is a named set of host variables that is defined in your program's DATA DIVISION. Host structures have a maximum of two levels, even though the host structure might itself occur within a multilevel structure. An exception is the declaration of a varying-length character string, which requires another level that must be level 49.

A host structure name can be a group name whose subordinate levels name elementary data items. For example:

```
01 A
  02 B
    03 C1 PICTURE ...
    03 C2 PICTURE ...
```

In this example, B is the name of a host structure consisting of the elementary items C1 and C2.

When writing an SQL statement using a qualified host variable name (for example, to identify a field within a structure), use the name of the structure followed by a period and the name of the field (that is, PL/I style). For example, specify STRUCTURE.FIELD rather than FIELD OF STRUCTURE or FIELD IN STRUCTURE. However, PL/I style applies only to qualified names within SQL statements; you cannot use this technique for writing qualified names in COBOL statements.

After the host structure is defined, you can refer to it in an SQL statement instead of listing the several host variables (that is, the names of the data items that comprise the host structure).

For example, you can retrieve all column values from selected rows of the table USER1.TEMPL with:

```
01 PEMPL.
    10 EMPNO              PIC X(6).
    10 FIRSTNME.
       49 FIRSTNME-LEN    PIC S9(4) USAGE COMP-4.
       49 FIRSTNME-TEXT   PIC X(12).
    10 MIDINIT            PIC X(1).
    10 LASTNAME.
       49 LASTNAME-LEN    PIC S9(4) USAGE COMP-4.
       49 LASTNAME-TEXT   PIC X(15).
    10 DEPTNO             PIC X(3).
  ...
  MOVE '000220' TO EMPNO.
  ...
  EXEC SQL
    SELECT *
      INTO :PEMPL
      FROM USER1.TEMPL
      WHERE EMPNO = :EMPNO
  END-EXEC.
```

Notice that in the declaration of PEMPL, two varying-length string elements are included in the structure: FIRSTNME and LASTNAME.

Basic Requirements for Host Variables

This section describes the coding rules for host variables, assignment rules, declarations for host variables, and indicator variables and structures as they apply to COBOL/400 programs.

Coding Rules for Host Variables

Precede a host variable with a colon when it is used within an SQL statement. Do not precede a host variable with a colon when you use it outside of an SQL statement.

You can use any valid host variable name in an SQL statement. You can use a host variable to represent a data value, but it cannot be used to represent a table name, view name, or column name.

SQL uses the COPY DD-format-name, COPY DD-ALL-FORMATS, COPY DDS-format-name, and COPY DDS-ALL-FORMATS to retrieve host variables from the file definitions. If the REPLACING option is specified, only complete name replacing is done. Identifier-1 is compared against the format name and the field name. If they are equal, identifier-2 is used as the new name.

Note: You cannot retrieve host variables from file definitions that have field names which are COBOL reserved words.

To retrieve the definition of the sample table TDEPT described in Appendix B, you can code the following:

```
01   TDEPT-STRUCTURE.
     COPY DDS-ALL-FORMATS OF TDEPT.
```

A host structure named TDEPT-STRUCTURE is defined with an 05 level field named TDEPT-RECORD that contains four 06 level fields named DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT. These field names can be used as host variables in SQL statements. For more information on the COBOL COPY verb, see the *COBOL/400 User's Guide*.

You can use any valid COBOL variable name for host variables. For example, the name can contain hyphens except as the first or last characters in the name.

You cannot specify JUSTIFIED or BLANK WITH ZERO when coding a host variable.

You can specify OCCURS only when defining an indicator structure. You cannot specify OCCURS for any other type of host variable.

Assignment Rules

SQL column values are sent to (or assigned to) host variables during the running of FETCH and SELECT statements. SQL column values are set from (or assigned from) host variables during the running of INSERT and UPDATE statements. All assignment operations observe the rules specified in the section "Assignment Rules" on page 4-2.

Allowable COBOL Declarations for Host Variables

SQL does not recognize every possible COBOL data description. If the COBOL data item you code is not consistent with those shown in the following table, the data item is not recognized as a host variable and might result in an error message when the program is precompiled. You can examine the data types in the cross-reference listing of the precompiler to note which variables have recognized data types.

The table shows the COBOL declarations that are allowed for use as host variables. All other types of COBOL declarations are rejected by SQL if encountered.

Note: Under *Explanatory Notes*, the length of string variables does not effect their recognition by the precompiler; the length limits apply to use, not declaration.

SQL Data Type	COBOL Equivalent	Explanatory Notes
CHAR (character string)	01 identifier PIC X(n). or 01 identifier PICTURE IS X(n).	**n** is a positive integer. Anything other than Xs in the PICTURE clause makes the data item unacceptable as a host variable.
(varying-length character string)	01 identifier 49 identifier PIC S9(ni) COMP-4. 49 identifier PIC X(nc).	**ni** is a positive integer from 1 to 4. **nc** is a positive integer.
(continued)		
SMALLINT (halfword integer)	01 identifier PIC S9(n) COMP-4. or 01 identifier PIC S9(n)V9(m) COMP-4. or 01 identifier PICTURE S9(n) USAGE IS COMPUTATIONAL-4.	**n** is a positive integer from 1 to 4. **m** also is a positive integer; **n+m** cannot exceed 4. You can include a **V** to denote the decimal point. Anything other than a 9 and a V makes the data item unacceptable as a host variable. PIC S and COMP-4 are required.
INTEGER (fullword integer)	01 identifier PIC S9(n) COMP-4. or 01 identifier PIC S9(n)V9(m) COMP-4. or 01 identifier PICTURE S9(n) USAGE IS COMPUTATIONAL-4.	**n** is a positive integer from 5 to 9. **m** also is a positive integer; **n+m** can range from 5 to 9. You can include a **V** to denote the decimal point. Anything other than a 9 and a V makes the data item unacceptable as a host variable. PIC S and COMP-4 are required.
DECIMAL (decimal value)	01 identifier PIC S9(n)V COMP-3. or 01 identifier PIC S9(n)V9(m) COMP-3. or 01 identifier PIC S9(n)V9(m) USAGE IS COMPUTATIONAL-3.	**n** and **m** are positive integers; **n+m** cannot exceed 18. You must include a **V** to denote the decimal point. Anything other than a 9 and a V makes the data item unacceptable as a host variable. COMP may be used in place of COMP-3. PIC S and COMP-3 or COMP are required.
NUMERIC (zoned decimal)	01 identifier PIC S9(n)V9(m)	**n** and **m** are positive integers; **n+m** cannot exceed 18. You must include a **V** to denote the decimal point. Anything other than a 9 and a V makes the data item unacceptable as a host variable. PIC S is required.

The following COBOL abbreviations are acceptable when coding COBOL host variables:

PIC	PICTURE or PICTURE IS
COMP	COMPUTATIONAL or USAGE IS COMPUTATIONAL
PIC IS	PICTURE IS
USAGE	USAGE IS
S9(4)	S9999
X(3)	XXX

Notes:

1. For data declarations of host variables not within a host structure, use level number 01 or 77. Host variables within a structure may be level 02 through 48. A leading zero is not required. Host structures have a maximum of two levels, except for level 49. Host structures may be used within levels 02 through 48.
2. A varying-length character string data item must have group level 01 through 48. The group must contain two elementary items with the level 49 and consists of:
   - Length item. The first elementary item must be a halfword integer variable (PICTURE S9(4) COMP-4. or PICTURE S9999 COMP-4.). It represents the length of the character string.
   - Value item. The second elementary item must have the same description as a fixed-length character string (for example, PICTURE X(80), where 80 is the maximum length of the string). This item is used to contain the value of the character string. If you use the host variable to insert a character string into a column, SQL inserts only as many characters as are allowed by the column length.

   The data item and elementary items can have any acceptable COBOL name. However, you refer to this host variable in your SQL statements with its group name only.
3. Any level 77 data description entry except the length item of a varying-length character string may be followed by one or more REDEFINES or RENAMES entries. However, the names in these entries may not be used in SQL statements. Entries with the name FILLER are ignored.
4. Arrays (OCCURS clause) are not supported other than as indicator variables for structures.
5. COBOL host variables used in SQL statements must be type compatible with the columns with which they are to be used:
   - Numeric data types are compatible with each other.
   - Character data types are compatible with each other. SQL automatically converts a fixed-length character string to a varying-length string, or vice versa, when necessary.
6. Be careful of overflow. For example, if you retrieve an INTEGER column value into a PICTURE S9(4) host variable and the column value is larger than 32767 or smaller than −32768, you get an overflow error.

7. Be careful of truncation. For example, if you retrieve an 80-character CHAR column value into a PICTURE X(70) host variable, the rightmost 10 characters of the retrieved string are truncated.

Indicator Variables and Structures

An indicator variable is a halfword integer variable used to indicate whether its associated host variable has been assigned a null value. See "Indicator Variables and Structures" on page 4-4 for more information.

Handling SQL Error Return Codes

When an SQL statement is processed in your program, SQL places a return code in the SQLCODE field of the SQL communication area (SQLCA). The SQLCA allows SQL to communicate with your program. For more information on handling error return codes, see "Handling SQL Error Return Codes" on page 4-6. For a description of SQLCA, see Appendix B. SQLCA and SQLDA Control Blocks in the *SQL/400 Reference* manual.

Chapter 6. Using SQL Statements in RPG Programs

RPG/400 supports both RPG II and RPG III programs. SQL statements can only be used in RPG III programs. RPG II and AutoReport are **not** supported. All references to RPG in this manual apply to RPG III only.

This chapter describes the unique application and coding requirements for embedding SQL statements in an RPG program. Requirements for host structures and host variables are defined, and the handling of return codes is described.

A detailed sample RPG program, showing how SQL statements can be used, is provided in Appendix D.

Application Requirements

To run SQL statements, your RPG III program must have an SQL communication area (SQLCA).

SQL Communication Area (SQLCA)

The SQL precompiler automatically places the SQLCA data structure into the source prior to the first calculation specification. You should not code an INCLUDE SQLCA in your RPG program. If the program has an INCLUDE SQLCA specified it is accepted and ignored. For more information on SQLCA, see Appendix B. SQLCA and SQLDA Control Blocks in the *SQL/400 Reference* manual.

The SQLCA, as defined for RPG:

```
ISQLCA       DS                                                  SQL
I                                        1   8 SQLAID            SQL
I                                      B 9  12 0SQLABC           SQL
I                                      B 13 16 0SQLCOD           SQL
I                                       17  86  SQLERM           SQL
I                                       87  94  SQLERP           SQL
I                                       95 118  SQLERR           SQL
I                                      B 95  98 0SQLER1          SQL
I                                      B 99 102 0SQLER2          SQL
I                                      B 103 106 0SQLER3         SQL
I                                      B 107 110 0SQLER4         SQL
I                                      B 111 114 0SQLER5         SQL
I                                      B 115 118 0SQLER6         SQL
I                                        119 119 SQLWN0          SQL
I                                        120 120 SQLWN1          SQL
I                                        121 121 SQLWN2          SQL
I                                        122 122 SQLWN3          SQL
I                                        123 123 SQLWN4          SQL
I                                        124 124 SQLWN5          SQL
I                                        125 125 SQLWN6          SQL
I                                        126 126 SQLWN7          SQL
I                                        127 137 SQLEXT          SQL
```

Note: Variable names in RPG are limited to 6 characters. The standard SQLCA names have been changed to a length of 6. RPG does not have a way of defining arrays in a data structure without also defining them in the extension

specification. SQLERR is defined as character with SQLER1 through 6 used as the names of the elements.

Coding Requirements

SQL statements coded in RPG programs must be placed in the calculation section. This requires that a C be placed in position 6. SQL statements can be placed in detail calculations, in total calculations, or in an RPG subroutine. The SQL statements are run based on the logic of the RPG statements.

An SQL statement is delimited by /EXEC SQL and /END-EXEC.

EXEC SQL indicates the beginning of a SQL statement. It must occupy positions 8 through 16 of the source statement, preceded by a / in position 7. The SQL statement may start in position 17 and continue through position 74.

END-EXEC ends the SQL statement. It must occupy positions 8 through 16 of the source statement, preceded by a / in position 7. Positions 17 through 74 must be blank.

Continuation for SQL Statements: When additional records are needed to contain the SQL statement, positions 9 through 74 can be used. Position 7 must be a + (plus sign), and position 8 must be blank.

Comments: RPG comments can be embedded within the SQL statement by placing an * in position 7.

The following is an example of an SQL statement in RPG:

Note: Both uppercase and lowercase letters are acceptable in SQL statements.

```
C           SWITCH    IFEQ ON
C                     MOVE 'E01'  HOSTV 3
C/EXEC SQL insert into TDEPT (DEPTNO, DEPTNAME, MGRNO, ADMRDEPT)
C+ values('A00', 'COMPUTER SERVICE DIV', '000010', :HOSTV)
C/END-EXEC
C                     END
```

The SQL statement inside the delimiters can be coded free-form and is not restricted to uppercase notation. The following example is the previous example coded differently:

```
C           SWITCH    IFEQ ON
C                     MOVE 'E01'  HOSTV 3
C/EXEC SQL
C+     INSERT into TDEPT  (DEPTNO,
C+                                 DEPTNAME,
C+                                   MGRNO,
C+                                      ADMRDEPT)
C*  this is a comment
C+         values ('A00',
C+                                'COMPUTER SERVICE DIV',
C+                                  '000010',
C+                                     :hostv)
C/END-EXEC
C                       END
```

Sequence Numbers: The source statements generated by the SQL precompiler are generated with the same sequence number as the SQL statement.

Reserved Words: Do not use host variable names that begin with SQL. In addition, a list of SQL reserved words can be found in Appendix A.

Dynamic SQL in a RPG Program: Dynamic SQL is an advanced programming technique described in Chapter 8. With dynamic SQL, your program can develop and then run SQL statements while the program is running. A SELECT statement with a variable SELECT list (that is, a list of the data to be returned as part of the query) that runs dynamically requires an SQL descriptor area (SQLDA), because you cannot know in advance how many or what type of variables to allocate in order to receive the results of the SELECT. Because the SQLDA uses pointer variables which are not supported by RPG, an INCLUDE SQLDA statement cannot be specified in an RPG program. An SQLDA must be set up by a PL/I program and passed to the RPG program in order to use it.

Using RPG Host Variables in SQL Statements: When your program retrieves data, the values are put into data items defined by your program and specified with the INTO clause of a SELECT or FETCH statement. The data items are called **host variables**. See "Using Host Variables in SQL Statements" on page 4-1 for more information on using host variables.

Host Structures

The RPG data structure name can be used as a **host structure** name if subfields exist in the data structure. The use of the data structure name in an SQL statement implies the list of subfield names making up the data structure.

When subfields are not present for the data structure, then the data structure name is a host variable of character type. This allows character variables larger than 256, because data structures can be up to 9999.

In the following example, BIGCHR is an RPG data structure without subfields. SQL treats any references to BIGCHR as a character string with a length of 642.

```
IBIGCHR      DS                  642
```

In the next example, PEMPL is the name of the host structure consisting of the subfields EMPNO, FIRSTNME, MIDINIT, LASTNAME, and DEPTNO. The reference to PEMPL uses the subfields. For example, the first column of TEMPL is placed in *EMPNO*, the second column is placed in *FIRSTNME*, and so on.

```
     IPEMPL       DS
     I                                        01  06 EMPNO
     I                                        07  18 FIRSTNME
     I                                        19  19 MIDINT
     I                                        20  34 LASTNAME
     I                                        35  37 DEPTNO

...
     C                    MOVE '000220'   EMPNO

...
     C/EXEC SQL
     C+ SELECT *  INTO :PEMPL
     C+ FROM  USER1.TEMPL
     C+ WHERE  EMPNO = :EMPNO
     C/END-EXEC
```

When writing an SQL statement, references to subfields can be qualified. Use the name of the data structure, followed by a period and the name of the subfield. For example, PEMPL.MIDINT is the same as specifying only MIDINT.

Basic Requirements for Host Variables

RPG has two types of data: character and numeric. Within the numeric type, there are four storage types: halfword binary, fullword binary, packed, and zoned. All variables defined in RPG can be used as host variables, except for the following:

- Multiple occurrence data structures
- Indicator field names (*INxx)
- Tables
- UDATE
- UDAY
- UMONTH
- UYEAR
- Look-ahead fields

Arrays cannot be used as host variables, but they can be used as indicator variables. Fields used as host variables are passed to SQL, using the CALL/PARM functions of RPG. If a field cannot be used in the results field of the PARM, it cannot be used as a host variable.

The following table shows the relationship between SQL and RPG data types:

SQL Data Type	RPG Equivalent
CHAR (Character string)	Field defined without decimal places; maximum length, 9999.
SMALLINT (halfword integer)	Subfield of the data structure. B in position 43 of the subfield specification, storage length, 2. An entry must be made in position 52.
INTEGER (fullword integer)	Subfield of the data structure. B in position 43 of the subfield specification, storage length, 4. An entry must be made in position 52.
DECIMAL (Decimal value)	Subfield of the data structure. P in position 43 and an entry in position 52 of the subfield specification. Defined as numeric and not a subfield of a data structure. Maximum precision of 30.
NUMERIC (Zoned decimal)	Subfield of the data structure. Blank in position 43. An entry must be made in position 52. Maximum precision of 30.

Coding Rules for Host Variables

Precede a host variable with a colon when it is used within an SQL statement. Do not precede a host variable with a colon when you use it outside of an SQL statement.

Any valid host variable name can be used in an SQL statement. You can use a host variable to represent a data value, but you cannot use it to represent a table name, view name, or column name.

Assignment Rules

RPG associates precision and scale with all numeric types. RPG defines numeric operations, assuming the data is in packed format. This means that operations involving binary variables include an implicit conversion to packed before the operation is performed (and back to binary, if necessary). Data is aligned to the implied decimal point when SQL operations are performed. All assignment operations observe the rules specified in the section "Assignment Rules" on page 4-2.

Indicator Variables and Structures

SQL requires the attribute of SMALLINT for data to be used as an indicator variable. This is accomplished in RPG by declaring the variable as a subfield in a data structure. The length for *to* and *from* must be 2. Position 43 must be B, and position 52 must be 0.

An indicator structure can be defined by declaring the variable as an array with an element length of 4,0 and declaring the array name as a subfield of a data structure with B in position 43.

The following example shows RPG field names used as indicator variables. The array AI has three elements defined as numeric 4,0. The entries for AI and IND as subfields declare their storage format of binary. AI can now be used as an indicator structure. IND can be used as an indicator variable.

```
E                    AI          3  4 0
.
.
.
I             DS
I                                      B  1  60AI
I                                      B  7  80IND
```

External Descriptions

The SQL precompiler processes the RPG source in much the same manner as the RPG compiler. This means that the precompiler processes the /COPY statement for definitions of host variables. Field definitions for externally described files are obtained and renamed, if different names are specified. The external definition form of the data structure can be used to obtain a copy of the column names to be used as host variables.

In the following example, the sample table TDEPT is used as a file in an RPG program. The SQL precompiler retrieves the field (column) definitions for TDEPT for use as host variables. The record name must be changed so that it is not the same as the file name. The field names must be changed because they are more than six characters in length.

```
FTDEPT     IP  E                    DISK
F                TDEPT                          KRENAMETDPTREC
ITDPTREC
I                  DEPTNAME                     DEPTN
I                  ADMRDEPT                     ADMRD
```

In the following example, the sample table is specified as an external data structure. The SQL precompiler retrieves the field (column) definitions as subfields of the data structure. Subfield names can be used as host variable names, and the data structure name TDEPT can be used as a host structure name. The field names must be changed because they are greater than six characters.

```
ITDEPT     E   DS
I                 DEPTNAME                      DEPTN
I                 ADMRDEPT                      ADMRD
```

The WHENEVER Statement

The target for the SQL GOTO statement must be the label of an RPG TAG statement. The RPG scope rules for the GOTO/TAG must be observed. For more information on handling the WHENEVER statement, see "Handling Exception Conditions with the WHENEVER Statement" on page 4-7.

Handling SQL Error Return Codes

When an SQL statement is processed in your program, SQL places a return code in the SQLCOD field of the SQL communication area (SQLCA). The SQLCA allows SQL to communicate with your program. For more information on handling error return codes, see "Handling SQL Error Return Codes" on page 4-6. For a description of SQLCA, see Appendix B. SQLCA and SQLDA Control Blocks in the *SQL/400 Reference* manual.

Chapter 7. Using SQL Statements in PL/I Programs

This chapter describes the unique application and coding requirements for embedding SQL statements in an AS/400 PL/I program. Requirements for host structures and host variables are defined. The handling of return codes is described.

A detailed sample PL/I program, showing how SQL statements can be used, is provided in Appendix D.

Application Requirements

To run SQL statements, your PL/I program must have an SQL communication area (SQLCA), as described below.

SQL Communication Area (SQLCA)

An AS/400 PL/I program that accesses SQL data must include an SQLCA. The SQLCA is the data area in which SQL returns information about the results of running each SQL statement. To include a PL/I declaration of the SQLCA, use the EXEC SQL INCLUDE statement:

```
EXEC SQL INCLUDE SQLCA;
```

When you use the EXEC SQL INCLUDE statement, SQL replaces that statement with a declaration of the SQLCA. The included PL/I source statements for the SQLCA are:

```
DCL 1 SQLCA,
      2 SQLCAID       CHAR(8),
      2 SQLCABC       BIN FIXED(31),
      2 SQLCODE       BIN FIXED(31),
      2 SQLERRM       CHAR(70) VAR,
      2 SQLERRP       CHAR(8),
      2 SQLERRD(6)    BIN FIXED(31),
      2 SQLWARN,
        3 SQLWARN0    CHAR(1),
        3 SQLWARN1    CHAR(1),
        3 SQLWARN2    CHAR(1),
        3 SQLWARN3    CHAR(1),
        3 SQLWARN4    CHAR(1),
        3 SQLWARN5    CHAR(1),
        3 SQLWARN6    CHAR(1),
        3 SQLWARN7    CHAR(1),
      2 SQLEXT        CHAR(8);
```

For more information on SQLCA, see Appendix B. SQLCA and SQLDA Control Blocks in the *SQL/400 Reference* manual.

SQL Descriptor Area (SQLDA)

Dynamic SQL is an advanced programming technique and is described in Chapter 8. With dynamic SQL, your program can develop and then run SQL statements while the program is running.

A SELECT statement with a variable SELECT list (that is, a list of the data to be returned as part of the query) that runs dynamically requires an SQL descriptor area (SQLDA), because you cannot know in advance how many or what type of variables to allocate in order to receive the results of the SELECT. The SQLDA is used to pass information about an SQL statement between SQL and your application. This information is used before an SQL statement is dynamically run. However, information about the results of running the statement is returned in the SQLCA. An SQLDA can be specified in a PL/I program by coding:

```
EXEC SQL INCLUDE SQLDA;
```

A PL/I program can also issue dynamic SQL statements using the EXECUTE IMMEDIATE statement, which does not require an SQLDA.

You may code the SQLDA structure directly instead of using the INCLUDE SQLDA statement. If you choose to declare the structure directly, you can specify any name for it. For example, you can call it SPACE1 or DAREA instead of SQLDA. The generated PL/I source statements for the SQLDA are:

```
DCL 1 SQLDA BASED(SQLDAPTR),
      2 SQLDAID       CHAR(8),
      2 SQLDABC       BIN FIXED(31),
      2 SQLN          BIN FIXED,
      2 SQLD          BIN FIXED,
      2 SQLVAR(99),
        3 SQLTYPE     BIN FIXED,
        3 SQLLEN      BIN FIXED,
        3 SQLRES      CHAR(12),
        3 SQLDATA     PTR,
        3 SQLIND      PTR,
        3 SQLNAME     CHAR(30) VAR;
DCL SQLDAPTR PTR;
```

Coding Requirements

You can code an SQL statement anywhere in a PL/I program.

When you issue SQL statements from a PL/I program, begin each statement with EXEC SQL. The remainder of the statement can appear on the next and subsequent lines.

Each SQL statement must have an ending delimiter. In PL/I programs, end each SQL statement with a semicolon (;). For example, an UPDATE statement issued from a PL/I program would look like this:

```
EXEC SQL
  UPDATE USER1.TDEPT
    SET MGRNO = :MGR_NUM
    WHERE DEPTNO = :INT_DEPT;
```

SQL statements, like PL/I statements, can have a label prefix.

Continuation for SQL Statements: The line continuation rules for SQL statements are the same as those for other PL/I statements.

Comments: You can include PL/I comments in SQL statements wherever you can have a blank, except between the words EXEC and SQL.

Margins: Code SQL statements within the margins specified by the MARGINS parameter on the CRTSQLPLI command. If EXEC SQL does not start within the specified margins, the SQL precompiler will not recognize the SQL statement. For more information about the CRTSQLPLI command, see the section "Precompiler Commands" on page 9-8.

Reserved Words: Host variable names that begin with SQL are reserved for SQL. A list of additional SQL reserved words is located in Appendix A.

PL/I %INCLUDE Directive: You cannot use the PL/I %INCLUDE directive to include source which contains SQL statements or host variable DCL statements. Use the SQL INCLUDE statement to include source which contains SQL statements and host variable DCL statements.

Using PL/I Host Variables in SQL Statements

When your program retrieves data, the values are put into data items defined by your program and specified with the INTO clause of a SELECT or FETCH statement. The data items are called **host variables**. See "Using Host Variables in SQL Statements" on page 4-1 for more information on using host variables.

Host Structures

In PL/I programs, you can define a **host structure**, which is a named collection of elementary data items. A host structure name can be a group name whose subordinate levels name elementary data items. For example:

```
DCL 1 A,
      2 B,
        3 C1 CHAR(...),
        3 C2 CHAR(...);
```

In this example, B is the name of a host structure consisting of the elementary items C1 and C2.

You can use the structure name as shorthand notation for a list of scalars, but only for a two-level structure. You can qualify a host variable with a structure name (for example, STRUCTURE.FIELD). Host structures are limited to two levels. (For example, in the above host structure example, the A cannot be referenced in SQL.) A structure cannot contain an intermediate level structure. In the previous example, A could not be used as a host variable or referenced in an SQL statement. A host structure for SQL data is two levels deep and can be thought of as a named collection of host variables. After the host structure is defined, you can refer to it in an SQL statement instead of listing the several host variables (that is, the names of the data items that make up the host structure).

For example, you can retrieve all column values from selected rows of the table USER1.TEMPL with:

```
DCL 1 PEMPL,
      5 EMPNO    CHAR(6),
      5 FIRSTNME CHAR(12) VAR,
      5 MIDINIT  CHAR(1),
      5 LASTNAME CHAR(15) VAR,
      5 DEPTNO CHAR(3);
 ...
EMPID = '000220';
 ...
  EXEC SQL
    SELECT *
      INTO :PEMPL
      FROM USER1.TEMPL
      WHERE EMPNO = :EMPNO;
```

Basic Requirements for Host Variables

This section describes the coding rules for host variables, assignment rules, declarations for host variables, and indicator variables and structures as they apply to PL/I programs.

Coding Rules for Host Variables

Precede a host variable with a colon when it is used within an SQL statement. Do not precede a host variable with a colon when you use it outside of an SQL statement.

Any valid host variable name can be used in an SQL statement. You can use a host variable to represent a data value, but you cannot use it to represent a table name, view name, or column name.

Host variables must be scalars or structures of scalars. Host variables cannot be declared as arrays, although an array of indicator variables is allowed when the array is associated with a host structure.

You can use any valid PL/I variable name for host variables.

You can declare host variable attributes in any order acceptable to PL/I. For example, BIN FIXED(31), BINARY FIXED(31), BIN(31) FIXED, and FIXED BIN(31) are all acceptable. If several variables have exactly the same attributes, you can declare all of them in a single declare statement. For example:

```
DCL (X, Y, Z) BIN FIXED;
```

The names of host variables should be unique within the program, even if the host variables are in different blocks or procedures, unless you qualify them with a structure name to make them unique.

If host variables are used in an SQL statement, the SQL statement must be within the scope of the DCL statement in which the variable was declared.

Host variables can be STATIC, BASED, or AUTOMATIC storage class, or parameters.

Assignment Rules

SQL column values are sent to (or assigned to) host variables during the running of FETCH and SELECT statements. SQL column values are set from (or assigned from) host variables during the running of INSERT and UPDATE statements. All assignment operations observe the rules specified in the section "Assignment Rules" on page 4-2.

Allowable PL/I Declarations for Host Variables

SQL will not recognize every possible PL/I data declaration. If the PL/I declaration you code is not consistent with those shown in the following table, the declaration is not recognized as a host variable and might result in an error message when the program is precompiled.

The following table shows the PL/I declarations that are allowed as host variables. All other types of PL/I declarations are ignored by SQL if encountered.

SQL Data Type	PL/I Equivalent	Explanatory Notes
CHAR (character string)	DCL identifier CHAR(m);	**m** is a positive integer from 1 to 32767.
(Varying length character string)	DCL identifier CHAR(n) VAR;	**n** is a positive integer from 1 to 32765.
SMALLINT (halfword integer)	DCL identifier BIN FIXED; or DCL identifier FIXED BIN; or DCL identifier BIN FIXED(15); or DCL identifier FIXED BIN(15);	
INTEGER (fullword integer)	DCL identifier BIN FIXED(31); or DCL identifier FIXED BIN(31);	
DECIMAL (decimal value)	DCL identifier DEC FIXED(p); or DCL identifier FIXED DEC(p); or DCL identifier DEC FIXED(p,s); or DCL identifier FIXED DEC(p,s);	**p** (the precision) and **s** (the scale factor) are positive integers. **s** is 0 to 15 but cannot be larger than **p**. **p** is 1 to 15.
NUMERIC	DCL identifier PICTURE '999V99R';	Valid picture data items are 9, V, and R; R is required. The range of numeric values in the picture specification is expressed by 9 in digit positions, V for the decimal, and R for the sign. For example, imagine the expression `PICTURE '999V99R',` where the length equals 6, and the precision equals 3. V is optional, R is required.
REAL (single-precision floating-point)	DCL identifier BIN FLOAT(n); or DCL identifier FLOAT BIN(n); or DCL identifier DEC FLOAT(m); or DCL identifier FLOAT DEC(m);	**n** is a positive integer from 1 to 24, however, SQL recognizes it as FLOAT(24). **m** is a positive integer from 1 to 7, however, SQL recognizes it as FLOAT(24).
FLOAT(53) (double-precision floating-point)	DCL identifier BIN FLOAT(n); or DCL identifier FLOAT BIN(n); or DCL identifier DEC FLOAT(m); or DCL identifier FLOAT DEC(m);	**n** is a positive integer from 25 to 53, however, SQL recognizes it as FLOAT(53). **m** is a positive integer from 8 to 16, however, SQL recognizes it as FLOAT(53).

The following PL/I abbreviations are acceptable when coding PL/I host variables:

BIN	BINARY
CHAR	CHARACTER
DEC	DECIMAL
DCL	DECLARE
PIC	PICTURE
VAR	VARYING

Notes:

1. You cannot declare:
   - DECIMAL without FIXED or FLOAT
   - BINARY without FIXED or FLOAT
   - BIT
   - Implicit declarations
   - Block scoping rules
   - Arrays—other than for indicators

   Otherwise, any valid PL/I DECLARE statement can be coded (as long as the declaration is within the parameters noted in the preceding table). If BASED is coded, it must be followed by a PL/I element-locator-expression.
2. PL/I host variables used in SQL statements must be type compatible with the columns with which they are to be used:
   - Numeric data types are compatible with each other.
   - Character data types are compatible with each other. SQL automatically converts a fixed-length character string to a varying-length string, or vice versa, when necessary.
3. Numeric data types are compatible with each other.
4. Be careful of overflow. For example, if you retrieve an INTEGER column value into a BIN FIXED(15) host variable and the column value is larger than 32767 or smaller than −32768, you get an overflow error.
5. Be careful of truncation. For example, if you retrieve an 80-character CHAR column value into a CHAR(70) host variable, the farthest right 10 characters of the retrieved string are truncated.

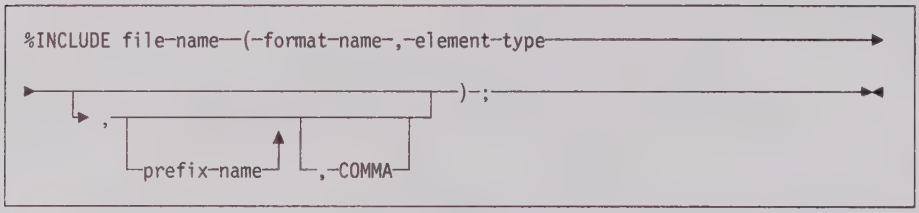
   Retrieving a floating-point numeric or decimal results column value into a BIN FIXED(31) or BIN FIXED(15) host variable removes any fractional value.

Indicator Variables and Structures

An indicator variable is a halfword integer variable used to indicate whether its associated host variable has been assigned a null value. See "Indicator Variables and Structures" on page 4-4 for more information.

Using the %INCLUDE Directive for External File Descriptions

You can use the PL/I %INCLUDE directive to include the definitions of externally described files in a source program. When used with SQL, only a particular format of the %INCLUDE directive is recognized by the SQL precompiler. That directive format must have the following three elements or parameter values, otherwise the precompiler ignores the directive. The required elements are *file name, format name,* and *element type*. There are two optional elements supported by the SQL precompiler: prefix name and COMMA. The syntax for the %INCLUDE directive and a description of the required and optional elements follows:



Element Description

A description of the required values follows:

file name
The name of the file that contains the record format to be included. You cannot name your file SYSLIB.

format name
The name of the record format to be included.

element type
The fields and indicators that are to be included. The valid types are:

INPUT
This type generates the record definition matching the input buffer. It includes fields that have a usage of INPUT or BOTH in the data description specification (DDS). OUTPUT fields are also included for subfiles.

OUTPUT
This type generates the record definition matching the output buffer. It includes fields that have a usage of OUTPUT or BOTH in DDS. For data base tables, it uses INPUT and BOTH.

KEY
This type includes fields specified as keys in DDS.

INDICATORS
This type is not supported by the AS/400 system SQL precompiler.

Note: If this element is present, the entire %INCLUDE directive is ignored by the precompiler.

RECORD
This type generates the record definition matching both the input and output buffers used by physical and logical files. It includes fields that have a usage of BOTH in DDS. INPUT fields are also included for logical files.

prefix name
A character string of 30 characters or less prefixing all generated names. The generated names must be valid and 31 characters or less.

COMMA
A value specifying that the last data element of the record or key structure does not end the structure. If COMMA is not specified (the default), the last data element ends the structure.

For more information on using the %INCLUDE directive, see the *PL/I User's Guide/Reference*.

Structure Definition

If the record format contains any of the fields of the element-type you specified, there are two factors that affect the names of the host variables that are generated: the *prefix* specified in the %INCLUDE directive, and the *alias* specified in the DDS source.

alias
If you use the ALIAS keyword in your DDS, the alias name you specify is the name the precompiler generates in your program. The field name specified in the DDS is ignored. By using the alias name, you can make full use of PL/Is 31-character name-length limit to increase the readability of your program.

prefix
If you code a prefix name in your %INCLUDE directive, the prefix you specify is attached to the field name or its alias supplied by the file. For example, if you write

```
%INCLUDE STOCK FILE(COUNT,RECORD,CURRENT_);
```

and, in the DDS, if the first field in the record is named

```
SALARY
```

the generated name is

```
CURRENT_SALARY
```

By using prefixes, you can generate meaningful names for different uses of the same record format inside a single program.

If one of the following conditions exists in the externally defined file, then the SQL precompiler generates a declaration of a host variable called DUMMDCL:

- No fields are defined for the file
- The element type is INPUT and no fields exist in the record format with usage of INPUT or BOTH
- The element type is OUTPUT, the file is not a data base file, and no fields exist in the record format with usage OUTPUT or BOTH
- The element type is INDICATORS, but no separate indicators exist

The DUMMYDCL host variable does not have a valid SQL data type and cannot be used. For example, a DUMMYDCL host variable is generated by a subfile control record format (which has no record fields and exists to define indicators and to communicate with the system).

A DUMMYDCL host variable is also generated if the record format you include has no fields of the element-type you specify. For example, if you specify INPUT

for a record format that has only output fields, no fields are included, and a DUMMYDCL is generated.

The following table shows how each DDS data type is defined in a PL/I program:

DDS Data Type	Length	Decimal Position	SQL Equivalent
Indicator	1	0	Not usable as a host variable
A	1 to 32766	none	CHAR(n) where n = 1 to 32766
B	1 to 4	0	SMALLINT
B	5 to 9	0	INTEGER
B	1 to 4	1 to 4	CHAR(2)
B	5 to 9	1 to 9	CHAR(4)
P	1 to 15	0 to 15	DECIMAL(p,q) where: p = 1 to 15 q = 0 to 15
P	16 to 31	0 to 31	CHAR(n) where n = length ÷ 2 + 1
S	1 to 15	0 to 15	NUMERIC(p,q) where: p = 1 to 15 q = 0 to 15
S	16 to 31	0 to 31	CHAR(n) where n = 16 to 31
F	1 to 7	0 to 7	FLOAT(24)
F	8 to 15	0 to 15	FLOAT(53)

Structure Ending

The structure is ended normally by the last data element of the record or key structure. However, if in the %INCLUDE directive the COMMA element is specified, then the structure is not ended.

For more information about %INCLUDE structures, see the *PL/I User's Guide/Reference*.

%INCLUDE Example

To include the definition of the sample table TDEPT described in Appendix B, you can code:

```
DCL 1 TDEPT_STRUCTURE,
%INCLUDE TDEPT(TDEPT,RECORD);
```

In the above example, a host structure named TDEPT_STRUCTURE would be defined having four fields. The fields would be DEPTNO, DEPTNAME, MGRNO, and ADMRDEPT.

Handling SQL Error Return Codes

When an SQL statement is processed in your program, it places a return code in the SQLCODE field of the SQL communication area (SQLCA). The SQLCA allows SQL to communicate with your program. For more information on handling error return codes, see "Handling SQL Error Return Codes" on page 4-6. For a description of SQLCA, see Appendix B. SQLCA and SQLDA Control Blocks in the *SQL/400 Reference* manual.

Chapter 8. Dynamic SQL Applications

Dynamic SQL allows an application to define and run SQL statements at program run time. An application that provides for dynamic SQL accepts as input (or builds) an SQL statement in the form of a character string. The application does not need to know what type of SQL statement it will run. The application:

- Builds or accepts as input an SQL statement
- Prepares the SQL statement for running
- Runs the statement
- Handles SQL return codes

Interactive SQL (described in Chapter 10) is an example of a dynamic SQL program. SQL statements are processed and run dynamically by interactive SQL.

Note: The run-time overhead is greater for statements processed using dynamic SQL than for static SQL statements. The additional process is similar to that required for precompiling, binding, and then running a program, instead of only running it. Therefore, only applications requiring the flexibility of dynamic SQL should use it. Other applications should access data from the data base using normal (static) SQL statements.

Some dynamic SQL statements require use of address variables. COBOL or RPG programs require the aid of PL/I programs to manage the address variables.

The examples in this chapter are PL/I examples. The following table shows all the statements supported by the SQL/400 and if they can be used in a dynamic application:

Note: In the following table, the numbers in the *Dynamic SQL* column correspond to the notes on the next page.

SQL Statement	Static SQL	Dynamic SQL
BEGIN DECLARE SECTION	Y	N
CLOSE	Y	N
COMMENT ON	Y	Y
COMMIT	Y	Y
CREATE DATABASE	Y	Y
CREATE INDEX	Y	Y
CREATE TABLE	Y	Y
CREATE VIEW	Y	Y
DECLARE CURSOR	Y	4
DECLARE STATEMENT	Y	N
DELETE	Y	Y
DESCRIBE	Y	6
DROP	Y	Y
END DECLARE SECTION	Y	N
EXECUTE	Y	1
EXECUTE IMMEDIATE	Y	2
FETCH	Y	N
GRANT	Y	Y
INCLUDE	Y	N
INSERT	Y	Y
LABEL ON	Y	Y
LOCK TABLE	Y	Y
OPEN	Y	N
PREPARE	Y	3
REVOKE	Y	Y
ROLLBACK	Y	Y
SELECT	Y	5
UPDATE	Y	Y
WHENEVER	Y	N

Notes:

1. Not used dynamically itself, but used to run prepared SQL statements. The SQL statement must be previously prepared by the PREPARE statement prior to using the EXECUTE statement. See example for PREPARE under "Using the PREPARE and EXECUTE Statements" on page 8-4.
2. Not used dynamically itself, but used with dynamic statement strings that do not have any ? parameter markers. The EXECUTE IMMEDIATE statement causes the statement strings to be prepared and run dynamically at program run time. See example for EXECUTE IMMEDIATE under "Processing NonSELECT Statements" on page 8-3.
3. Can not be prepared, but used to parse, optimize, and set up dynamic SELECT statements prior to running. See example for PREPARE under "Processing NonSELECT Statements" on page 8-3.
4. Not used dynamically itself, but used to define the cursor for the associated dynamic SELECT statement prior to running.
5. A SELECT statement cannot be used with EXECUTE or EXECUTE IMMEDIATE, but can be prepared and used with OPEN CURSOR.
6. Cannot be prepared, but used to return a description of a prepared statement.

Designing and Running a Dynamic SQL Application

To issue a dynamic SQL statement, you must use the statement with either an EXECUTE statement or an EXECUTE IMMEDIATE statement, because dynamic SQL statements are not prepared at precompile time and therefore must be prepared at run time. The EXECUTE IMMEDIATE statement causes the SQL statement to be prepared and run dynamically at program run time.

There are two basic types of dynamic SQL statements: SELECT and nonSELECT statements. NonSELECT statements include such statements as DELETE, INSERT, and UPDATE.

Processing NonSELECT Statements

To build a dynamic SQL nonSELECT statement:

1. Verify that the SQL statement you want to build is one that can be run dynamically (see the preceding table).
2. Build the SQL statement. (Use Interactive SQL for an easy way to build, verify, and run your SQL statement. See Chapter 10 for more information.)

To run a dynamic SQL nonSELECT statement:

1. Run the SQL statement using EXECUTE IMMEDIATE.
2. Handle any SQL return codes that might result.

The following is an example of an application running a dynamic SQL nonSELECT statement (stmtstrg):

```
EXEC SQL
  EXECUTE IMMEDIATE :stmtstrg;
```

Note: The SQL statement is normally a host variable. In PL/I, it can also be a string expression.

Using the PREPARE and EXECUTE Statements

If nonSELECT statements contain no parameters markers, they can be run dynamically using the EXECUTE IMMEDIATE statement. However, if the nonSELECT statements have parameter markers, they must be run using PREPARE and EXECUTE.

The PREPARE statement prepares the nonSELECT statement (for example, the DELETE statement) and gives it a name of your choosing, in this instance, let us call it S1. After the statement has been prepared, it can be run many times within the same program, using different values for the parameter markers. The following example is of a prepared statement being run multiple times:

```
DSTRING = 'DELETE FROM USER1.TEMPL WHERE EMPNO = ?';

/*The ? is a parameter marker which denotes
  that this value is a host variable that is
  to be substituted each time the statement is run.*/

EXEC SQL PREPARE S1 FROM :DSTRING;

/*DSTRING is the delete statement that the PREPARE statement is
  naming S1.*/

 DO UNTIL (EMP =0);
/*The application program reads a value for EMP from the
  display station.*/
   EXEC SQL
      EXECUTE S1 USING :EMP;


 END;
```

In routines similar to the example above, you must know the number of parameters and their data types, because the host variables that provide the input data are declared when the program is being written.

Note: All SQL statements that have been prepared are destroyed when COMMIT or ROLLBACK is run, unless the HOLD is specified. For more information, see descriptions of COMMIT and ROLLBACK in the *SQL/400 Reference* manual.

Processing SELECT Statements and Using SQLDA

There are two basic types of SELECT statements: **fixed-list** and **varying-list**. To process a fixed-list SELECT statement, an SQLDA is not necessary.

To process a varying-list SELECT statement, you must first declare a SQLDA structure. SQLDA is a control block used to pass host variable input values from an application program to SQL and to receive output values from SQL. In addition, information about parameter markers and output host variables or SELECT columns can be returned in a PREPARE or DESCRIBE statement.

Fixed-List SELECT Statements

In dynamic SQL, fixed-list SELECT statements are those statements designed to retrieve data of a predictable number and type. When using these statements, you can anticipate and define host variables to accommodate the retrieved data, so that an SQLDA is not necessary. Each successive FETCH returns the same number of values as the last, and these values have the same data formats as those returned for the last FETCH. You can specify host variables the same as you would for any SQL application.

You can use fixed-list dynamic SELECT statements with any SQL-supported application program.

To run fixed-list SELECT statements dynamically, your application must:

1. Place the input SQL statement into a host variable.
2. Issue a PREPARE statement to validate the dynamic SQL statement and put it into a form that can be run.
3. Declare a cursor for the statement name.
4. Open the cursor.
5. FETCH a row into a fixed list of variables (rather than into a descriptor area, as you would if you were using a varying-list SELECT statement, described in the following section, "Varying-List SELECT Statements")
6. When end of data occurs, close the cursor.
7. Handle any SQL return codes that may result.

For example,

```
MOVE = 'SELECT EMP, EMPNAME
  FROM USER1.TEMPL WHERE EMP> ?'
  TO DSTRING.

EXEC SQL
  PREPARE S2 FROM :DSTRING END-EXEC.

EXEC SQL
  DECLARE C2 CURSOR FOR S2 END-EXEC.

EXEC SQL
  OPEN C2 USING :EMP END-EXEC.

DO WHILE (SQLCODE = 0);

EXEC SQL
  FETCH C2 INTO :EMP, :EMPNAME END-EXEC.

END;

EXEC SQL
  CLOSE C2 END-EXEC.
```

Note: Remember that because the SELECT statement, in this case, always returns the same number and type of data items as previously run fixed-list SELECT statements, you do not have to use the SQL descriptor area (SQLDA). The preceding example can be run with COBOL/400, AS/400 PL/I, and RPG III (part of RPG/400).

Varying-List SELECT Statements

In dynamic SQL, varying-list SELECT statements are ones for which the number and format of data items to be returned are not predictable; that is, you do not know how many variables you need, or what the data types are. Therefore, you cannot define host variables in advance to accommodate the data items returned.

If your application accepts varying-list SELECT statements, your program has to:

1. Place the input SQL statement into a host variable.
2. Issue a PREPARE statement, to validate the dynamic SQL statement and put it into a form that can be run.
3. Declare a cursor for the statement name.
4. Issue a DESCRIBE statement, to request information from the data base about the type and size of each column of the results table.

   Notes:

   a. You can also code the PREPARE statement with an INTO clause to perform the functions of PREPARE and DESCRIBE with a single statement.

   b. If the SQLDA is not large enough to contain column descriptions for each retrieved column, the program must determine how much space is needed, get storage for that amount of space, build a new SQLDA, and reissue the DESCRIBE statement.

5. Allocate the amount of storage needed to contain a row of retrieved data.
6. Put storage addresses into the SQLDA (SQL descriptor area), to tell the data base where to put each item of retrieved data.
7. Open the cursor (declared in step 3) that includes the name of the dynamic SELECT statement.
8. FETCH a row.
9. When end of data occurs, close the cursor.
10. Handle any SQL return codes that might result.

The SQL Descriptor Area (SQLDA)

You can use the SQLDA to pass information about an SQL statement between the data base and your application.

The SQLDA is a collection of variables required for running the SQL DESCRIBE statement, and may optionally be used by the PREPARE, OPEN, FETCH, and EXECUTE statements. An SQLDA communicates with dynamic SQL. It can be used in a DESCRIBE statement, changed with the addresses of host variables, and then reused in a FETCH statement.

The meaning of the information in an SQLDA depends on its use. In PREPARE and DESCRIBE, an SQLDA provides information to an application program about a prepared statement. In OPEN, EXECUTE, and FETCH, an SQLDA provides information about host variables.

If your application lets you have several cursors open at the same time, you can code several SQLDAs, one for each dynamic SELECT statement. For more information on SQLDA and SQLCA, see the *SQL/400 Reference* manual.

SQLDAs can be used in RPG and COBOL applications as well as PL/I; however, because RPG and COBOL provide no way to set pointers, pointers must be set outside the RPG or COBOL program by a PL/I program.

SQLDA Format

The SQLDA consists of four variables followed by an arbitrary number of occurrences of a sequence of five variables collectively named SQLVAR. When an SQLDA is used in OPEN, FETCH, and EXECUTE, each occurrence of SQLVAR describes a host variable.

The variables of SQLDA are as follows:

SQLDAID — SQLDAID is an eye-catcher for storage dumps. It is a string of 8 characters that have the value 'SQLDA' after the SQLDA is used in a PREPARE or DESCRIBE statement. It is not used for FETCH, OPEN, or EXECUTE.

SQLDABC — SQLABC indicates the length of the SQLDA. It is a 4-byte integer that has the value SQLN*LENGTH(SQLVAR) + 16 after the SQLDA is used in a PREPARE or DESCRIBE statement. SQLDABC must have a value equal to or greater than SQLN*LENGTH(SQLVAR) + 16 prior to use by FETCH, OPEN, or EXECUTE.

SQLN — SQLN is a 2-byte integer that specifies the total number of occurrences of SQLVAR.

SQLD — SQLD is a 2-byte integer that specifies the pertinent number of occurrences of SQLVAR, that is, the number of host variables described by the SQLDA.

SQLVAR — The variables of SQLVAR are SQLTYPE, SQLLEN, SQLRES, SQLDATA, SQLIND, and SQLNAME. These variables are defined as follows:

SQLTYPE — SQLTYPE is a 2-byte integer that specifies the data type of the host variable as follows:

Values	Data Type	Null Indicator? (for each value)
500/501	2-byte integer	No/Yes
496/497	4-byte integer	No/Yes
484/485	decimal	No/Yes
488/489	numeric (zoned)	No/Yes
480/481	4-byte and 8-byte floating point (IEEE)	No/Yes
452/453	fixed-length character	No/Yes
448/449	varying-length character	No/Yes
456/457	long varying-length character (more than 254 bytes)	No/Yes

SQLLEN — SQLLEN is a 2-byte integer variable that is always pertinent, based on the following conditions:

- If the data type is decimal or numeric, the first byte is the precision and the second byte is the scale.
- If the data type is fixed-length character or floating point, SQLLEN is the length of the host variable or column.
- If the data type is varying-length character, SQLLEN must be from 1 to 254 bytes, which indicates the maximum length of the host variable.
- If the data type is long varying-length character, SQLLEN must be from 1 to 32 766 bytes, which indicates the maximum length of the host variable.
- If the data type is a 2-byte or 4-byte integer, SQLLEN is either the length of the host variable or column or precision and scale. If the first byte is 0, SQLLEN contains a length of 2 or 4. If the first byte is nonzero, the first byte is the precision and the second byte is the scale.

SQLRES — SQLRES is a 12-byte reserved area for boundary alignment purposes. Note that pointers *must* be on a quad-word boundary.

SQLDATA

SQLDATA is a 16-byte pointer variable that specifies the address of the host variables when the SQLDA is used on OPEN, FETCH, and EXECUTE. When the SQLDA is used on PREPARE and DESCRIBE, this area is overlaid with the following information:

- The first 4 bytes contain two small integers and indicate the SBCS character set and code page for a character column. The character set and code page is determined by the system value QCHRID, which is set by the user and contains both the SBCS character set and code page identification. If the integer is zero, either the column is not character type, or the character is bit data (FOR BIT DATA).

SQLIND

SQLIND is a 16-byte pointer that specifies the address of a small integer host variable that is used as an indication of null or not null. A negative value indicates null and a nonnegative indicates not null. Since SQL/400 does not provide full null value support, this has limited usefulness. There are cases, however, when the result of a query is a null value even though nulls cannot be stored in tables. These cases are:

- If a group function (MIN, MAX, and so on) is specified in the SELECT list, and the GROUP BY clause is not specified, and the result of COUNT is 0, then a null value (−1) is returned in the indicator variable for the group functions other than COUNT.
- If a decimal data error occurred when evaluating an expression in the SELECT list, but a successful analysis could still be made as to whether the resulting row should be selected, as many valid results are returned as possible, and items that encountered errors are returned as a null value (−2).

When the SQLDA is used on PREPARE and DESCRIBE, this area is overlaid with the following information:

- The first 4 bytes contains an integer. If the column is not a character column that contains both SBCS and DBCS data (FOR MIXED DATA), the value of this field is 0. Otherwise the integer is not equal to 0.

SQLNAME

SQLNAME is a variable-length character variable of maximum length 30 which contains the name of selected column or label names after a PREPARE or DESCRIBE. It is not used in OPEN, FETCH, or EXECUTE.

Example of a SELECT Statement for Allocating Storage for SQLDA

The SELECT statement can be read from a display station or from a host variable, or it can be developed within an application program. The following example shows a SELECT statement read from a display station:

```
SELECT DEPTNO, PHONENO FROM USER1.TEMPL
  WHERE LASTNAME = 'PARKER'
```

Note: The SELECT statement has no INTO clause. Dynamic SELECT statements must *not* have an INTO clause, even if they return only one row.

When the statement is read, it is assigned to a host variable. The host variable (for example, named QSTRING) is then processed, using the PREPARE statement, as shown:

```
EXEC SQL
  PREPARE S1 FROM :QSTRING;
```

Allocating Storage

Now you can allocate storage for the SQLDA. The techniques for acquiring storage are language dependent. The SQLDA must be allocated on a 16-byte boundary. The SQLDA consists of a fixed-length header, 16 bytes long. The header is followed by a varying-length array section (SQLVAR), each element of which is 80 bytes in length. The amount of storage you need to allocate depends on how many elements you want to have in the SQLVAR array. Each column you select must have a corresponding SQLVAR array element. Therefore, the number of columns listed in your select statement determines how many SQLVAR array elements you should allocate. Because SELECT statements are specified at run time, however, it is impossible to know how many columns will be accessed. Consequently, you must estimate the number of columns. Suppose, in this example, that no more than 20 columns are ever expected to be accessed by a single SELECT statement. This means that the SQLVAR array should have a dimension of 20 (for an SQLDA size 20 x 80, or 1600, plus 16 for a total of 1616 bytes), because each item in the select-list must have a corresponding entry in SQLVAR.

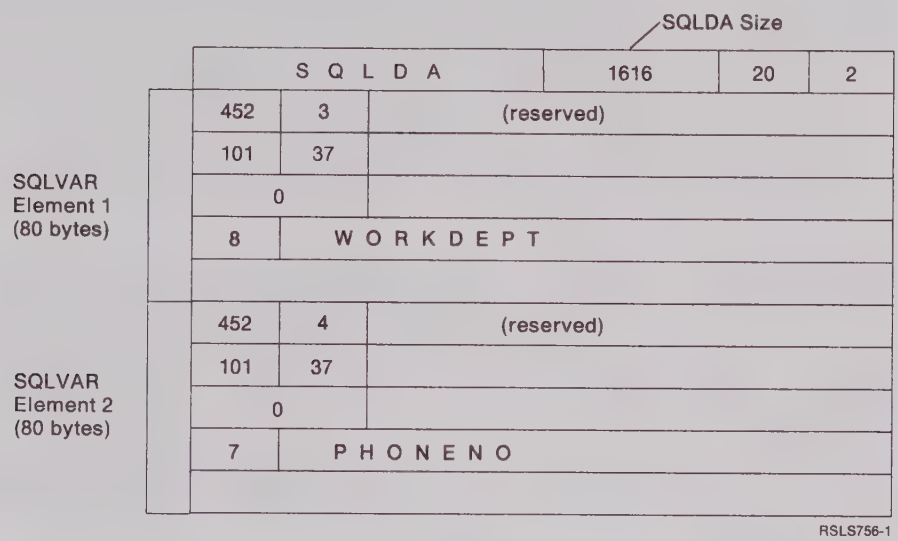
Having allocated what you estimated to be enough space for your SQLDA, in the SQLN field of the SQLDA, set an initial value equal to the number of SQLVAR array elements. In the following example, set SQLN to 20:

```
Allocate space for an SQLDA of 1616 bytes on a quadword boundary
Set SQLN = 20
```

Having allocated storage, you can now issue a DESCRIBE statement.

```
EXEC SQL
  DESCRIBE S1 INTO :SQLDA;
```

When the DESCRIBE statement is run, SQL places values in the SQLDA that provide information about the select-list. The following figure shows the contents of the SQLDA after the DESCRIBE is run:



SQLDAID is an identifier field initialized by SQL when a DESCRIBE is run. SQLDABC is the byte count or size of the SQLDA. You can ignore these for now.

The example for running the SELECT statement for S1 is:

```
SELECT DEPTNO, PHONENO
  FROM USER1.TEMPL
  WHERE LASTNAME = 'PARKER'
```

Your program might have to alter the SQLN value if the SQLDA is not large enough to contain the described SQLVAR elements. For example, let the SELECT statement contain 27 select-list expressions instead of the 20 or less that you estimated. Because the SQLDA was only allocated with a SQLVAR dimension of 20 elements, SQL cannot describe the select-list, because the SQLVAR has too many elements. SQL sets the SQLD to the actual number of columns specified by the SELECT statement, and the rest of the structure is ignored. Therefore, after a DESCRIBE, you should compare the SQLN to the SQLD. If the value of SQLD is less than or equal to the value of SQLN, allocate a larger SQLDA based on the value in SQLD, as follows:

```
EXEC SQL
  DESCRIBE S1 INTO :SQLDA;
IF SQLN <= SQLD THEN
DO;

  /*Allocate a larger SQLDA using the value of SQLD.*/
  /*Reset SQLN to the larger value.*/

EXEC SQL
  DESCRIBE S1 INTO :SQLDA;
END;
```

If you use DESCRIBE on a non-SELECT statement, SQL sets SQLD to 0. Thus if your program is designed to process both SELECT and nonSELECT statements, you can describe each statement (after it is prepared) to determine whether it is a SELECT statement. This sample routine is designed to process only SELECT statements; the SQLD is not checked.

Your program must now analyze the elements of SQLVAR. Remember that each element describes a single select-list expression. Consider, again, the SELECT statement that is being processed:

```
SELECT DEPTNO, PHONENO
  FROM USER1.TEMPL
  WHERE LASTNAME = 'PARKER'
```

The first item in the select-list is DEPTNO. At the beginning of this section, we identified that each SQLVAR element contains the fields SQLTYPE, SQLLEN, SQLDATA, SQLIND, and SQLNAME. SQL returns, in the SQLTYPE field, a code that describes the data type of the expressions and whether nulls are applicable or not.

For example, SQL sets SQLTYPE to 452 in SQLVAR element 1 (see Figure 8-1 on page 8-11). This specifies that DEPTNO is a fixed-length character string (CHAR) column and that nulls are not permitted in the column.

SQL sets SQLLEN to the length of the column. For character strings, SQLLEN is set to the maximum length of the character string. For decimal, numeric, or nonzero scale binary, the precision and scale are returned in the first and second bytes, respectively. For other data types, SQLLEN is set as follows:

```
SMALLINT -- SQLLEN = 2
INTEGER  -- SQLLEN = 4
REAL     -- SQLLEN = 4
FLOAT    -- SQLLEN = 8
```

Because the data type of DEPTNO is CHAR, SQL sets SQLLEN equal to the length of the character string. For DEPTNO, that length is 3. Therefore, when the SELECT statement is later run, a storage area large enough to hold a CHAR(3) string is needed.

Because the data type of DEPTNO is CHAR, the first 4 bytes of SQL data indicated the character set and code page. In this example, the character set is 101 and the code page is 37 (see Figure 8-1 on page 8-11). The last field in an SQLVAR element is a varying-length character string called SQLNAME. The first 2 bytes of SQLNAME contain the length of the character data. The character data itself is usually the name of a column used in the select statement (DEPTNO in the above example). The exceptions to this are select-list items that are unnamed, such as functions (for example, SUM(SALARY)), expressions (for example, A+B−C), and constants. In these cases, SQLNAME is an empty string. SQLNAME may also contain a label rather than a name. One of the parameters associated with the PREPARE and DESCRIBE statements is the USING clause. You can specify it this way:

```
EXEC SQL
  DESCRIBE S1 INTO SQLDA
    USING LABELS;
```

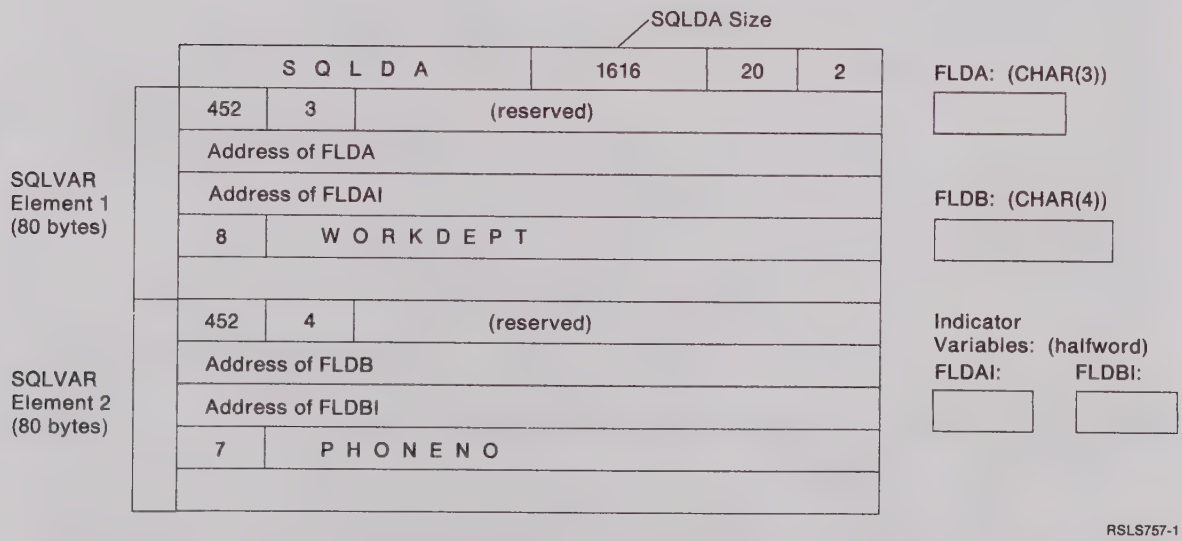
If you specify NAMES (or omit the USING parameter entirely), only column names are placed in the SQLNAME field. If you specify LABELS, only labels associated

with the columns listed in your SQL statement are entered here. If you specify ANY, labels are placed in the SQLNAME field for those columns that have labels; otherwise, the column names are entered. If you specify BOTH, names and labels are both placed in the field with their corresponding lengths. If you specify BOTH, however, you must remember to double the size of the SQLVAR array because you are including twice the number of elements. For more information on the USING option and on column labels, see the *SQL/400 Reference* manual.

In the example, the second SQLVAR element contains the information for the second column used in the select: PHONENO. The 452 code in SQLTYPE specifies that PHONENO is a CHAR column. For a CHAR data type of length 4, SQL sets SQLLEN to 4.

After analyzing the result of the DESCRIBE, you can allocate storage for variables containing the result of the SELECT statement. For DEPTNO, a character field of length 3 must be allocated; for PHONENO, a character field of length 4.

After the storage is allocated, you must set SQLDATA and SQLIND to point to the appropriate areas. For each element of the SQLVAR array, SQLDATA points to the place where the results are to be put. SQLIND points to the place where the null indicator is to be put. The following figure shows what the structure looks like now:



This is what was done so far:

```
EXEC SQL
  INCLUDE SQLDA;
/*Read a statement into the DSTRING varying-length
  character string host variable.*/
EXEC SQL
  PREPARE S1 FROM :QSTRING;
/*Allocate an SQLDA of 1616 bytes.*/
SQLN =20;
EXEC SQL
  DESCRIBE S1 INTO :SQLDA;
/*Analyze the results of the DESCRIBE.*/
/*Allocate storage to hold one row of
  the results table.*/
/*Set SQLDATA and SQLIND for each column
  of the results table.*/
```

Using a Cursor

You are now ready to retrieve the SELECT statements results. Dynamically defined SELECT statements must not have an INTO statement. Therefore, all dynamically defined SELECT statements must use a cursor. Special forms of the DECLARE, OPEN, and FETCH are used for dynamically defined SELECT statements.

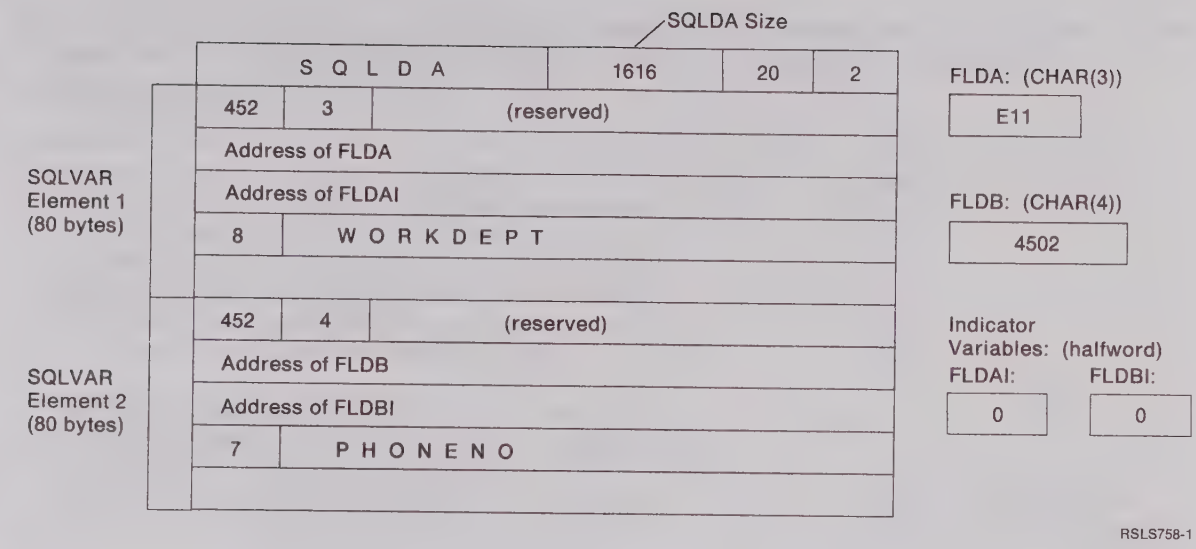
The DECLARE statement for the example statement is:

```
EXEC SQL DECLARE C1 CURSOR FOR S1;
```

As you can see, the only difference is that the name of the prepared SELECT statement (S1) is used instead of the SELECT statement itself. The actual retrieval of results rows is made as follows:

```
EXEC SQL
  OPEN C1;
EXEC SQL
  FETCH C1 USING DESCRIPTOR :SQLDA;
DO WHILE (SQLCODE = 0);
  /*Display ... the results pointed to by SQLDATA*/
  END
  /*Display ('END OF LIST')*/
EXEC SQL
  CLOSE C1;
```

The cursor is opened, and the results table is evaluated. Notice that there are no input host variables needed for the example SELECT statement. The SELECT results row are then returned using FETCH. On the FETCH statement, there is no list of output host variables. Rather, the FETCH statement tells SQL to return results into areas described by the descriptor called SQLDA. The same SQLDA that was set up by DESCRIBE is now being used for the output of the SELECT statement. In particular, the results are returned into the storage areas pointed to by the SQLDATA and SQLIND fields of the SQLVAR elements. The following figure shows what the structure looks like after the FETCH statement has been processed.



The meaning of the SMALLINT pointed to by SQLIND is the same as any other indicator variable:

```
 0  Denotes that the returned value is not null.
<0  Denotes that the returned value is null.
>0  Denotes that the returned value was truncated because
    the storage area furnished was not large enough.
    The indicator variable contains the length before
    truncation.
```

Note: Unless HOLD is specified, dynamic cursors are closed and prepared statements are destroyed during COMMIT or ROLLBACK. Before opening the cursor, you must issue a PREPARE to use the statement again, if HOLD was not specified on COMMIT or ROLLBACK.

Using Parameter Markers

In the example we are using, the SELECT statement that was dynamically run had predictable parameters (input host variables) in the WHERE clause. In the example, it was:

```
WHERE LASTNAME = 'PARKER'
```

If you want to run the same SELECT statement several times, using different values for LASTNAME, you can use an SQL statement such as PREPARE or EXECUTE, as described in "Using the PREPARE and EXECUTE Statements" on page 8-4, like this:

```
SELECT DEPTNO, PHONENO FROM USER1.TEMPL WHERE LASTNAME = ?
```

When your parameters are not predictable, your application cannot know the number or types of the parameters until run time. You can arrange to receive this information at the time your application is run, and, by using a USING DESCRIPTOR on the OPEN CURSOR statement, you can substitute the values contained in specific host variables for the parameter markers included in the WHERE clause of the SELECT statement.

To code such a program, you need to use the OPEN CURSOR statement with the USING DESCRIPTOR clause. This SQL statement is used to not only open a cursor, but to replace each parameter marker with the value of the corresponding host variable. The descriptor name that you specify with this statement must identify an SQLDA that contains a valid description of those host variables. This SQLDA, unlike those previously described, is not used to return information on data items that are part of a SELECT list. That is, it is not used as output from a DESCRIBE statement, but as input to the OPEN CURSOR statement. It provides information on host variables that are used to replace parameter markers in the WHERE clause of the SELECT statement. It gets this information from the application, which must be designed to place appropriate values into the necessary fields of the SQLDA. The SQLDA is then ready to be used as a source of information for SQL in the process of replacing parameter markers with host variable data.

When you use the SQLDA for input to the OPEN CURSOR statement with the USING DESCRIPTOR clause, not all of its fields have to be filled in. Specifically, SQLDAID and SQLNAME can be left blank. Therefore, when you used this method for replacing parameter markers with host variable values, you need to determine:

- How many ? parameter markers are there?
- What are the data types and attributes of these parameters markers (SQLTYPE and SQLLEN)?
- Do you want an indicator variable?

In addition, if the routine is to handle both SELECT and nonSELECT statements, you may want to determine what category of statement it is. (Alternatively, you can write code to look for the SELECT keyword.)

If your application uses parameter markers, your program has to:

1. Read a statement into the DSTRING varying-length character string host variable.
2. Determine the number of ? parameter markers.
3. Allocate an SQLDA of that size.
4. Set SQLN and SQLD to the number of ? parameter markers.
5. Set SQLDABC equal to SQLN*LENGTH(SQLVAR) + 16.
6. For each ? parameter marker:
    a. Determine the data types, lengths, and indicators.
    b. Set SQLTYPE and SQLLEN.
    c. Allocate storage to hold the input values (the ? values).
    d. Set these values.
    e. Set SQLDATA and SQLIND (if applicable) for each ? parameter marker.
    f. Issue the OPEN CURSOR statement with a USING DESCRIPTOR clause to open your cursor and substitute a host variable value for each of the parameter markers.

The statement can then be processed normally.

Chapter 9. Preparing and Running a Program with SQL Statements

This chapter describes some of the tasks of preparing and running an application program. The tasks described are:

- Precompiling
- Compiling
- Binding
- Running

Basic Processes of the SQL Precompiler

You must precompile and compile an application program containing embedded SQL statements before you can run it. Precompiling of such programs is done by the SQL precompiler. The SQL precompiler scans each statement of the application program source and does the following:

- **Looks for SQL statements and for the definition of host variable names.** The variable names and definitions are used to verify the SQL statements. You can examine the listing after the SQL precompiler completes processing to see if any errors occurred.
- **Verifies that each SQL statement is valid and free of syntax errors.** The validation procedure supplies error messages in the output listing that helps you correct any errors that occur.
- **Validates the SQL statements using the description in the data base.** During the precompile, the SQL statements are checked for valid table, view, and column names. If a referred to table or view does not exist at the time of the precompile or compile, the validation is done at run time. If the table or view does not exist at run time, an error occurs.

  Notes:

  1. Overrides are processed when retrieving external definitions. For more information, see the *Data Base Guide* and the *Data Management Guide*.
  2. You need some authority (at least *OBJOPR) to any tables or views referred to in the SQL statements in order to validate the SQL statements. The actual authority required to process any SQL statement is checked at run time. For more information on any SQL statement, see the *SQL/400 Reference* manual.
- **Prepares each SQL statement for compilation in the host language.** For most SQL statements, the SQL precompiler inserts a comment and a CALL statement to the SQL interface module (QSQROUTE). For some SQL statements (for example, DECLARE statements), the SQL precompiler produces no host language statement except a comment.
- **Produces information about each precompiled SQL statement.** The information is stored internally in a temporary source file member, where it is available for use during the bind process.

To get complete diagnostic information when you precompile, specify the *SOURCE and *XREF precompiler options.

Input to the Precompiler

Application programming statements and embedded SQL statements are the primary input to the SQL precompiler. In PL/I programs, the SQL statements must use the same margins as specified in the MARGINS parameter of the CRTSQLPLI command.

The SQL precompiler assumes that the host language statements are syntactically correct. If the host language statements are not syntactically correct, the precompiler may not correctly identify SQL statements and host variable declarations. There are limits on the forms of source statements that can be passed through the pecompiler. Literals and comments, that are not accepted by the application language compiler, can interfere with the precompiler source scanning process and cause errors.

The SQL INCLUDE statement can be used to get secondary input from the file specified by the INCFILE parameter of the CRTSQLxxx[1] command. The SQL INCLUDE statement causes input to be read from the specified member until the end of the member is reached. The included member may not contain other precompiler INCLUDE statements, but can contain both application program and SQL statements.

Another preprocessor may process source statements before the SQL precompiler. However, any preprocessor run before the SQL precompile must be able to pass through SQL statements.

If double-byte character set (DBCS) literals are specified in the application program source, the system value QIGC (for using DBCS characters) must indicate that the system supports DBCS literals.

Output from the Precompiler

The following sections describe the various kinds of output supplied by the precompiler.

Listing

The output listing is sent to the print file specified by the PRTFILE parameter of the CRTSQLxxx command. The following items are output to the print file:

- Precompiler options

  Options specified in the CRTSQLxxx command.

- Precompiler source

  This output supplies precompiler source statements, with record numbers assigned by the precompiler, if the *SOURCE option is in effect.

- Precompiler cross-reference

  This output supplies a cross-reference listing (if *XREF was specified in the OPTION parameter), showing the precompiler line numbers of SQL statements in which host names and column names are referred to.

- Precompiler diagnostics

---

[1] The xxx in this command refers to the host language indicators: CBL for COBOL/400, PLI for AS/400 PL/I, and RPG for RPG III.

This output supplies diagnostic messages, showing the precompiler record numbers of statements in error.

Temporary Source File Members

Source statements processed by the precompiler are written to QSQLTEMP in the QTEMP library. In your precompiler-changed source code, SQL statements have been converted to comments and calls to the SQL interface module, QSQROUTE, application language interface. The name of the temporary source file member is the same as the name specified in the PGM parameter of the CRTSQLxxx. This member cannot be changed before being used as input to the compiler.

QSQLTEMP can be moved to a permanent library after the precompile, if you want to compile at a later time. You can not change the records of the source member, or the attempted compile will fail.

Sample Precompiler Output

The output can provide information about your precompiled source module if you specify the *SOURCE (*SRC) and *XREF options on the OPTION parameter when you call the SQL precompiler.

The format of the precompiler output is:

```
5728ST1  R01  M02  YYMMDD                    IBM SQL/400       CTEST1                  12/18/87 14:35:52  Page 1

A  Source type..............COBOL
   Program name.............CLR/CBLTEST1
   Source file..............*LIBL/QCBLSRC
   Member...................CTEST1
   Precompile options.......*SRC      *XREF     *SQL
   INCLUDE file.............*LIBL/*SRCFILE
   Commit...................*CHG
   Generation level.........0
   Printer file.............*LIBL/QSYSPRT
   Text.....................*SRCMBRTXT

   Source member changed on 12/18/87  14:35:43
                            B
```

RSLS760-1

A A list of the options you specified when the SQL precompiler was called.

B The date the source member was last changed.

```
A  5728ST1  R01  M02  YYMMDD                IBM SQL/400       CTEST1                    B      C  12/18/87 14:35:52  Page 2
Record *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8    SEQNBR  Last change
     1         IDENTIFICATION DIVISION.                                                        100    12/18/87
     2         PROGRAM-ID.   EXAMPLE1.                                                         200    12/18/87
     3         ENVIROMENT DIVISION.                                                            300    12/18/87
     4         CONFIGURATION SECTION.                                                          400    12/18/87
     5         SOURCE-COMPUTER.  IBM-AS400                                                     500    12/18/87
     6         OBJECT-COMPUTER.  IBM-AS400                                                     600    12/18/87
     7         INPUT-OUTPUT SECTION.                                                           700    12/18/87
     8         FILE-CONTROL.                                                                   800    12/18/87
     9             SELECT OUTFILE, ASSIGN TO PRINTER-QPRINT,                                   900    12/18/87
    10             FILE STATUS IS FSTAT.                                                      1000    12/18/87
    11         DATA DIVISION.                                                                 1100    12/18/87
    12         FILE SECTION.                                                                  1200    12/18/87
    13         FD  OUTFILE                                                                    1300    12/18/87
    14             DATA RECORD IS REC-1,                                                      1400    12/18/87
    15             LABEL RECORD IS OMITTED.                                                   1500    12/18/87
    16         01  REC-1.                                                                    1600    12/18/87
    17             05 CC                          PIC X.                                      1700    12/18/87
    18             05 DEPT-NO                     PIC X(3).                                   1800    12/18/87
    19             05 FILLER                      PIC X(5)                                    1900    12/18/87
    20             05 AVERAGE-EDUCATION-LEVEL     PIC ZZZ.                                    2000    12/18/87
    21             05 FILLER                      PIC X(5).                                   2100    12/18/87
    22             05 AVERAGE-SALARY              PIC ZZZZ9.99.                               2200    12/18/87
    23         01  ERROR RECORD.                                                             2300    12/18/87
    24             05 CC                          PIC X.                                      2400    12/18/87
    25             05 ERROR-CODE                  PIC S9(5).                                  2500    12/18/87
    26             05 ERROR-MESSAGE               PIC X(70).                                  2600    12/18/87
    27         WORKING-STRORAGE SECTION.                                                      2700    12/18/87
    28             EXEC SQL                                                                   2800    12/18/87
    29               INCLUDE SQLCA                                                            2900    12/18/87
    30             END EXEC.                                                                  3000    12/18/87
    31         77  FSTAT                         PIC XX.                                     3100    12/18/87
    32         01  AVG-RECORD.                                                               3200    12/18/87
    33             05 DEPTNO                      PIC X(3).                                   3300    12/18/87
    34             05  AVG-EDUC                   PIC S9(4) USAGE COMP-4.                     3400    12/18/87
    35             05  AVG-SALARY                 PIC S9(6)V99 USAGE COMP-3.                  3500    12/18/87
    36         PROCEDURE DIVISION.                                                            3600    12/18/87
    37        ****************************************************************              3700    12/18/87
    38        *  This program will get the average education level and the   *              3800    12/18/87
    39        *  average salary by department.                               *              3900    12/18/87
    40        ****************************************************************              4000    12/18/87
    41         A000-MAIN-PROCEDURE.                                                           4100    12/18/87
    42             OPEN OUTPUT OUTFILE.                                                       4200    12/18/87
    43        ****************************************************************              4300    12/18/87
    44        *  Set-up WHENEVER statement to handle SQL errors              *              4400    12/18/87
    45        ****************************************************************              4500    12/18/87
    46             EXEC SQL                                                                   4600    12/18/87
    47               WHENEVER SQLERROR GOTO B000-SQL-ERROR                                    4700    12/18/87
    48             END-EXEC.                                                                  4800    12/18/87
    49        ****************************************************************              4900    12/18/87
    50        *  Declare cursor                                              *              5000    12/18/87
    51        ****************************************************************              5100    12/18/87
    52             EXEC SQL                                                                   5200    12/18/87
    53               DECLARE CURS CURSOR FOR                                                  5300    12/18/87
    54                 SELECT DEPTNO,AVG(EDUCLVL),AVG(SALARY)                               5400    12/18/87
    55                 FROM USER1.TEMPL                                                     5500    12/18/87
    56                 GROUP BY DEPTNO                                                      5600    12/18/87
    57             END-EXEC.                                                                  5700    12/18/87
    58        ****************************************************************              5800    12/18/87
    59        *  Open cursor                                                 *              5900    12/18/87
    60        ****************************************************************              6000    12/18/87
```

RSLS761-1

```
61          EXEC SQL                                                    6100    12/18/87
62            OPEN CURS                                                 6200    12/18/87
63          END-EXEC.                                                   6300    12/18/87
64      ****************************************************************6400    12/18/87
65      *  Fetch all result rows                                       *6500    12/18/87
66      ****************************************************************6600    12/18/87
67          PERFORM-A010-FETCH-PROCEDURE THROUGH A010-FETCH-EXIT        6700    12/18/87
68            UNTIL SQLCODE IS = 100.                                   6800    12/18/87
69      ****************************************************************6900    12/18/87
70      *  Close cursor                                                *7000    12/18/87
71      ****************************************************************7100    12/18/87
72          EXEC SQL                                                    7200    12/18/87
73            CLOSE CURS                                                7300    12/18/87
74          END-EXEC.                                                   7400    12/18/87
75          CLOSE OUTFILE.                                              7500    12/18/87
76          STOP RUN.                                                   7600    12/18/87
77      ****************************************************************7700    12/18/87
78      *  Fetch a row and move the information to the output record   *7800    12/18/87
79      ****************************************************************7900    12/18/87
80       A010-FETCH-PROCEDURE.                                          8000    12/18/87
81          MOVE SPACES TO REC-1.                                       8100    12/18/87
82          EXEC SQL                                                    8200    12/18/87
83            FETCH CURS INTO :AVG-RECORD                               8300    12/18/87
84          END-EXEC.                                                   8400    12/18/87
85          IF SQLCODE IS = 0                                           8500    12/18/87
86            MOVE DEPTNO TO DEPT-NO                                    8600    12/18/87
87            MOVE AVG-SALARY TO AVERAGE-SALARY                         8700    12/18/87
88            MOVE AVG-EDUC TO AVERAGE-EDUCATION-LEVEL                  8800    12/18/87
89            WRITE REC-1 AFTER ADVANCING 1 LINE.                       8900    12/18/87
90       A010-FETCH-EXIT.                                               9000    12/18/87
91          EXIT.                                                       9100    12/18/87
92      ****************************************************************9200    12/18/87
93      *  An SQL error occurred.  Move the error number to the error  *9300    12/18/87
94      *  record and stop running.                                    *9400    12/18/87
95      ****************************************************************6000    12/18/87
96       B000-SQL-ERROR.                                                9600    12/18/87
97          MOVE SPACES TO ERROR-RECORD.                                9700    12/18/87
98          MOVE SQLCODE TO ERROR-CODE.                                 9800    12/18/87
99          MOVE "AN SQL ERROR HAS OCCURRED" TO ERROR-MESSAGE.          9900    12/18/87
100         WRITE ERROR-RECORD AFTER ADVANCING 1 LINE.                 10000    12/18/87
101         CLOSE OUTFILE.                                             10100    12/18/87
102         STOP RUN.                                                  10200    12/18/87
                    ***** END OF SOURCE *****                          10200    12/18/87
```

RSLS762-1

A The record number assigned by the precompiler when it reads the source record. The record numbers are used to identify the source record in error messages and SQL runtime processing.

B The sequence number taken from the source record. This is the number seen when you use the source entry utility (SEU) to edit the source member.

C The date when the source record was last changed. If the record is blank, it indicates that the record has not been changed since it was created.

```
5728ST1  R01  M02  YYMMDD                IBM SQL/400      CTEST1                12/18/87 14:35:52  Page 3

                                          CROSS REFERENCE
A                                  B      C
Data Names                         Define Reference

'CURS'                             53     CURSOR                                  D
                                          62 73 83                                E
'DEPTNO'                           ****   COLUMN
                                          54 56
'EDUCLVL'                          ****   COLUMN
                                          54
'SALARY'                           ****   COLUMN
                                          54
'TEMPL'                            ****   TABLE IN 'USER1'
                                          55
'USER1'                            ****   AUTHID
                                          55
AVERAGE-EDUCATION-LEVEL            20     IN REC-1

AVERAGE-SALARY                     22     IN REC-1                                F                 G

AVG-EDUC                           34     SMALL INTEGER PRECISION(4,0) IN AVG-RECORD
AVG-RECORD                         32     STRUCTURE
                                          83
AVG-SALARY                         35     DECIMAL(8,2) IN AVG-RECORD

B000-SQL-ERROR                     ****   LABEL
                                          47
CC                                 17     CHARACTER(1) IN REC-1

CC                                 24     CHARACTER(1) IN ERROR-RECORD

DEPTNO                             18     CHARACTER(3) IN REC-1

DEPTNO                             33     CHARACTER(3) IN AVG-RECORD

ERROR-CODE                         25     NUMERIC(5,0) IN ERROR-RECORD

ERROR-MESSAGE                      26     CHARACTER(70) IN ERROR-RECORD

ERROR-RECORD                       23     STRUCTURE

FSTAT                              31     CHARACTER(2)

REC-1                              16
```

RSLS763-1

A Data names are the symbolic names used in source statements. Names enclosed in quotation marks (") or apostrophes (') are names of SQL entities, such as tables and columns. Names not enclosed by quotation marks or apostrophes are host variables.

B The define column specifies the line number at which the name is defined. The line number is generated by the SQL precompiler. **** means that the object was not defined or the precompiler did not recognize the declarations.

C The reference column contains two types of information:

- What the symbolic name is defined as **D**
- The line number(s) where the symbolic name occurs **E**

If the symbolic name refers to a valid host variable, the data type **F** or structure **G** is also noted.

```
5728ST1  R01  M02  YYMMDD                IBM SQL/400       CTEST1              12/18/87 14:35:52  Page 4

                                    DIAGNOSTIC MESSAGES

MSG ID  SEV  RECORD  TEXT
SQL1103   0      55  Position 21 Field definitions for file USER1 in QSYS not found.
SQL1103   0      55  Position 21 Field definitions for file TEMPL in USER1 not found.


                                                Message Summary
Total   Info    Warning    Error     Severe    Terminal
    2      2          0        0          0           0

 102 Source records processed

                          *****  E N D  O F  L I S T I N G  *****
```

RSLS764-1

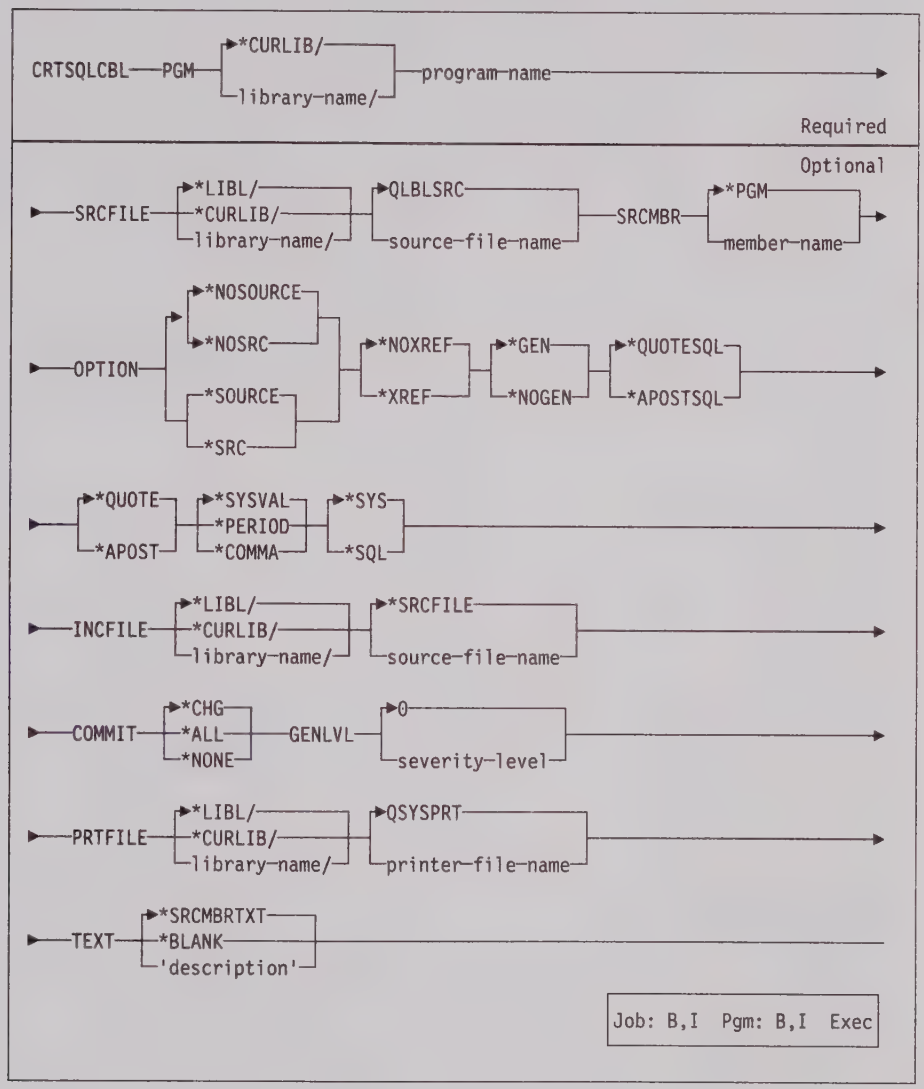
Precompiler Commands

In SQL/400 there are three precompiler commands (CRTSQLCBL, CRTSQLPLI, and CRTSQLRPG); one for each of the three host languages: COBOL/400, AS/400 PL/I, and RPG III (part of RPG/400). Separate commands by language let you specify the required parameters and then take the default for the remaining parameters, because the defaults are applicable only to the one language you are using. For example, the options *APOST and *QUOTE are unique to COBOL. They are not included in the commands for the other languages.

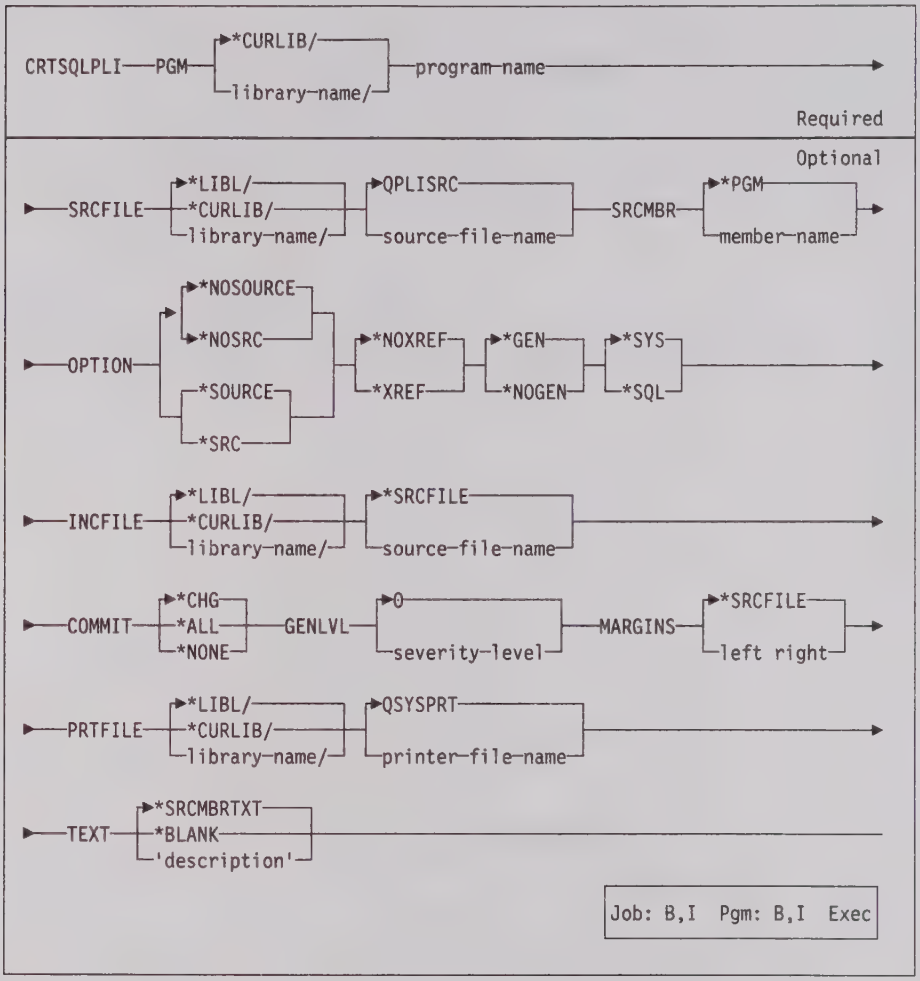
Syntax for the Precompiler Commands

The syntax for the SQL/400 precompiler commands is shown on the following pages. The parameters are defined under "Parameter Definitions" on page 9-12.

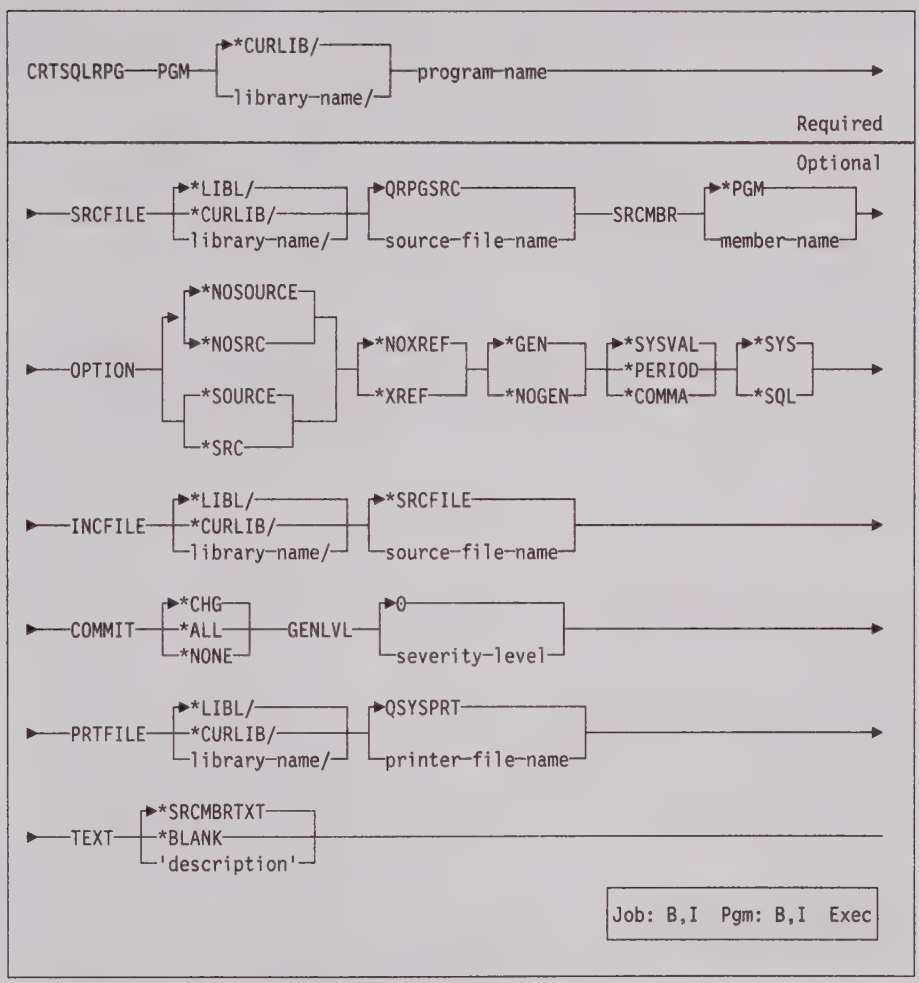
The syntax for the create SQL COBOL (CRTSQLCBL) precompiler command is:



The syntax for the create SQL PL/I (CRTSQLPLI) precompiler command is:



The syntax for the create SQL RPG (CRTSQLRPG) precompiler command is:



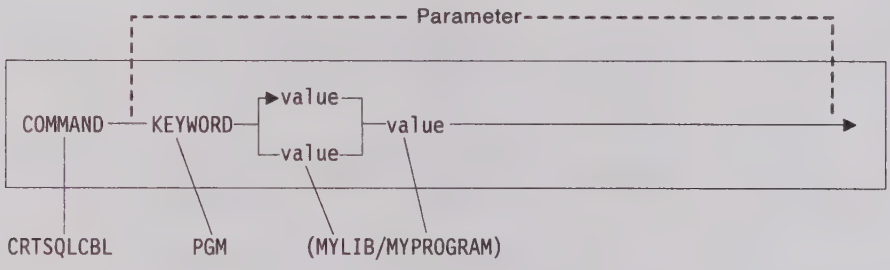
Precompiler Command Parameters

The precompiler is controlled by parameters specified when it is called by one of the SQL precompiler commands. The parameters specify how the input is processed, and how the output is presented.

You can precompile a program without specifying anything more than the name of the member containing the program source statements as the PGM parameter of the CRTSQLxxx. SQL assigns default values for all precompiler parameters (which may, however, be overridden by any that you explicitly specify).

Parameter Definitions

The following paragraphs describe the parameters for the precompiler commands. The parameters consist of **keywords** and **values**. The following figure defines the parts of a parameter as used throughout this manual:



Note: The default value is indicated by an arrowhead (→).

RSLS759-2

Note: The default value is indicated by an arrowhead.

Required Parameter

The following parameter is required for the SQL precompiler commands. If you choose not to specify any of the following keyword values, then the defaults for those values are used.

PGM (Program)

Specifies the qualified name by which the compiled program is known.

*CURLIB: If a library is not specified, the program is created in the current library. If no current library entry exists in the library list, QGPL is used.

library name: Specify the name of the library where the compiled program is created.

Warning: If the program name you specify is the same name of an existing program, your new program replaces the existing one.

program name: Specify the name of the program being created that contains the SQL statement(s).

Optional Parameters

The following parameters are optional for the SQL precompiler source commands. If you choose not to specify any of the following keywords or their values, then the defaults are used.

SRCFILE (Source File)

Specifies the qualified name of the source file that contains the source with the SQL statements.

***LIBL:** Specifies the library list used to locate the source file.

***CURLIB:** Specifies the current library for the job used to locate the source file. If no current library entry exists in the library list, QGPL is used.

library name: Specify the library where the source file is located.

QxxxSRC: If the source file name is not specified, the IBM-supplied source file name is used. QLBLSRC is the source file name for COBOL, QPLISRC for PL/I, and QRPGSRC for RPG.

source file name: Specify the name of the source file that contains the source. This source file should have a record length of 92. The source file can be a data base file, device file, or an inline data file.

SRCMBR (Source Member)

Specifies the name of the source file member containing the source. This parameter is only specified if the source file name in the SRCFILE parameter is a data base file.

***PGM:** Specifies that the host program source is in the member of the source file that has the same member name as that specified in the PGM parameter for the precompiler command.

member name: Specify the name of the member containing the host program source.

OPTION

Specifies the following options to the precompiler. If an option is specified more than once, or if two options conflict, the last option specified is used. If an option is not specified, then the default is used.

***NOSOURCE or *NOSRC:** Specifies that a source listing is not produced by the precompiler.

***SOURCE or *SRC:** Specifies that a source listing is produced by the precompiler, consisting of the source input and all error messages.

***NOXREF:** Specifies that the precompiler does not produce a cross-reference of names.

***XREF:** Specifies that the precompiler produce a cross-reference between items in your program and the numbers of the statements in your program that refer to these items.

***GEN:** Specifies that the host language compiler is to be called to create a program after the source has been precompiled.

***NOGEN:** Specifies that the host language compiler is not called and that no program will be created.

***QUOTESQL (COBOL only):** Specifies that the string delimiters within SQL statements are quotation marks (").

***APOSTSQL (COBOL only):** Specifies that the string delimiters within SQL statements are apostrophes (').

Notes:

1. If *APOSTSQL is specified, the SQL escape character for delimited identifiers is the quotation (") mark. If *QUOTESQL is specified, the SQL escape character for delimited identifiers is the apostrophe (').
2. SQL statements in RPG and PL/I use apostrophes for string delimiters and quotation marks for SQL escape characters.

***QUOTE (COBOL only):** Specifies that a quotation mark (") is used for non-numeric literals and Boolean literals in COBOL statements.

***APOST (COBOL only):** Specifies that an apostrophe (') is used for non-numeric literals and Boolean literals in COBOL statements.

***SYSVAL (COBOL and RPG only):** Specifies that the value used as the decimal point is from the QDECFMT system value.

***PERIOD (COBOL and RPG only):** Specifies that the value used as the decimal point is a period.

***COMMA (COBOL and RPG only):** Specifies that the value used as the decimal point is a comma.

Note: In PL/I, the period is used as the decimal point.

***SYS:** Specifies that the AS/400 system naming convention will be used, (library-name/file-name).

***SQL:** Specifies that the SQL naming convention will be used (data-base-name.table-name).

Note: *SYS also specifies that the AS/400 security convention will be used. *SQL specifies that the SQL security convention will be used. The AS/400 security convention requires that the user running the program must have authority to objects referred to by the SQL statements in the program. The SQL security convention requires that the owner of the program has authority to the objects referred to in the static SQL statements in the program. The SQL security convention is the same as the AS/400 security convention for dynamic SQL and interactive SQL.

INCFILE (Include File)

Specifies the qualified name of the source file that contains the members included in the program with any SQL INCLUDE statement.

***LIBL:** Specifies that the library list is used to locate the source file.

***CURLIB:** The current library for the job used to locate the source file. If no current library entry exists in the library list, QGPL is used.

library name: Specifies the library where the source file is located.

***SRCFILE:** Specifies the qualified source file you specified in the SRCFILE parameter that contains the source file member(s) specified on any SQL INCLUDE statement.

source file name: Specify the name of the source file that contains the source file member(s) specified on any SQL INCLUDE statement. The record length of the source file you specify here must be no less than the record length of the source file you specified for the SRCFILE parameter.

COMMIT

Specifies if SQL statements in the compiled program are run under commitment control. Files referred to in the host language source are not affected by this option. Only files, SQL tables, and SQL views referred to in SQL statements are affected.

***CHG (change):** Specifies that only the updated, deleted, and inserted rows are locked until the end of the unit of recovery (transaction).

***ALL:** Specifies that all rows selected, updated, deleted, and inserted are locked until the end of the unit of recovery (transaction).

***NONE:** Specifies that commitment control is not used. If SQL DDL statements are included in the program, *NONE must be used.

Note: If *CHG or *ALL are specified, the following data definition statements cannot be included in the application:

COMMENT ON	DROP
CREATE DATABASE	GRANT
CREATE INDEX	LABEL ON
CREATE TABLE	REVOKE
CREATE VIEW	

GENLVL (generation level)

Specifies if the CL Create Program command is called, depending on the severity of messages generated as a result of precompile errors. If errors occur with a severity level greater than the value specified in GENLVL parameter, the appropriate language compiler is not called.

0: If a severity level value is not specified, the default severity level is zero.
severity level Specify a number from 0 through 40. Some suggested values are listed below:

10 The level value for warnings.
20 The level value for general error messages.
30 The level value for serious error messages.
40 The level value for system detected error messages.

MARGINS (PL/I only)

Specifies the part of the precompiler input record that contains source text.

***SRCFILE:** Specifies that the margin values of the file member you specify in the SRCMBR parameter are used. If the file is of type PL/I, the margin values are the values specified on the Source Entry Utility display. If the file is a different type, the margin values are the default values of 2 and 72.

left right: Specify the beginning position (**left**) and the ending position (**right**) for the statements. The margins you specify must not be less than 1 or more than 80. The left margin must be smaller than the right margin.

PRTFILE (printer file)

Specifies the qualified name of the printer device file to which the precompiler listing is directed. The file should have a minimum length of 132. If a file with a record length of less than 132 is specified, information is lost.

***LIBL:** Specifies the library list used to locate the printer file.

***CURLIB:** Specifies the current library for the job used to locate the printer file. If no current library entry exists in the library list, QGPL is used.

library name: Specify the library where the printer file is located.

QSYSPRT: If a file name is not specified, the precompiler listing is directed to the IBM-supplied print file, QSYSPRT.

printer file name: Specify the name of the printer device file to which the precompiler listing is directed.

TEXT

Specifies text that briefly describes the program and its function.

***SRCMBRTXT:** Specifies that the text is taken from the source file member being used to create the program. You can add to or change the text for a data base source member by using the Start Source Entry Utility (STRSEU) command, or by using either the Add Physical File Member (ADDPFM) or Change Physical File Member (CHGPFM) command. If the source file is an inline file or a device file, the text is blank.

***BLANK:** Specifies no text.

description: Specify no more than 50 characters enclosed in apostrophes (').

Example of the Precompiler Source Command

The following example creates a COBOL program named SAMPLE and stores it in a library named QTEMP. The SQL naming convention was selected, and every row selected from the specified table(s) are locked until the end of the unit of recovery.

```
CRTSQLCBL PGM(qtemp/sample) OPTION(*SRC *XREF *SQL) COMMIT(*ALL)
```

Compiling an Application Program

SQL/400 automatically calls the host language compiler after the successful completion of a precompile, unless *NOGEN is specified. The CRTxxxPGM command is run specifying the program name, source file name, precompiler created source member name, text, and USRPRF. For COBOL, the *QUOTE or *APOST is also passed. For PL/I, the MARGINS are set in the temporary source file. Defaults are used for all other parameters.

Note: If you specify *SYS, USRPRF(*USER) is specified by the precompiler. If you specify *SQL, USRPRF(*OWNER) is specified by the precompiler.

You can interrupt the call to the host language compiler by specifying *NOGEN under OPTION parameter of the precompiler command. *NOGEN specifies that the host language compiler will not be called. The precompiler has created the source member in the QTEMP/QSQLTEMP file, using the program name specified in the CRTSQLxxx command as the name of the member. You now can explicitly call the host language compilers, specifying the source member in the QTEMP/QSQLTEMP file, and change the defaults, if you wish.

If you precompiled using the *SQL option, then you must specify USRPRF(*OWNER) on the CRTxxxPGM command.

Note: You must not change the source member in QTEMP/QSQLTEMP prior to issuing the CRTxxxPGM command or the compile will fail.

Warning: If you do separate precompile and compile steps, and the source program contains references to externally described files (COPY DDS in COBOL, %INCLUDE in PL/I, and externally defined files or data structures in RPG), the referred to files must not be changed between the precompile and compile steps. Otherwise, results that are not predictable may occur because the change to the field definitions are not reflected in the temporary source member.

Error and Warning Messages during a Compile

The conditions described in the following paragraphs could produce an error or warning message during an attempted compile process.

During a PL/I Compile: If EXEC SQL starts before the left margin (as specified with the MARGINS parameter, the default), the SQL precompiler will not recognize the statement as an SQL statement. Consequently, it will be passed as is to the compiler.

During a COBOL Compile: If EXEC SQL starts before column 12, the SQL precompiler will not recognize the statement as an SQL statement. Consequently, it will be passed as is to the compiler.

Binding an Application

Before you can run your application program, a relationship between the program and any referred to tables and views must be established. This process is called **binding**. The result of binding is an **access plan**.

The access plan is a control structure that describes the actions necessary to satisfy each SQL request. An access plan contains information about the program and about the data the program intends to use.

The access plan is stored in the program and therefore is automatically deleted, moved, saved, and so on with the program.

SQL/400 automatically attempts to bind and create access plans when a successful compile has occurred. If, at run time, the data base manager detects that an access plan is not valid or detects that changes have occurred to the data base that may improve performance (for example, the addition of indexes), a new access plan is automatically created. Binding does three things:

1. **It revalidates the SQL statements using the description in the data base.** During the bind process, the SQL statements are checked for valid table, view, and column names. If a referred to table or view does not exist at the time of the precompile or compile, the validation is done at run time. If the table or view does not exist at run time, a negative SQLCODE is returned.
2. **It selects the access paths needed to access the data your program wants to process.** In selecting an access path, indexes, table sizes, and other factors are considered, when it builds an access plan. It considers all indexes available to access the data and decides which ones (if any) to use when selecting a path to the data.
3. **It attempts to build access plans.** If all the SQL statements are valid, the bind process then builds and stores access plans in the program.

If the characteristics of a table or view your program accesses have changed, the access plan may no longer be valid. When you attempt to run a program that con-

tains an access plan that is not valid, the system automatically attempts to rebuild the access plan. If the access plan cannot be rebuilt, a negative SQLCODE is returned. In this case, you might have to change the program's SQL statements and reissue the CRTSQLxxx command to correct the situation.

For example, if a program contains a SQL statement that references COLUMNA in TABLEA and the user deletes and recreates TABLEA, such that COLUMNA no longer exists, and then calls the program, the automatic rebind will be unsuccessful because COLUMNA no longer exists. In this case the user must change the program source and reissue the CRTSQLxxx command.

Program References

All references to data bases, tables, views, and indexes in an SQL program are placed in the object information repository (OIR) of the library when the program is created.

You may use the Display Program References (DSPPGMREF[2]) CL command to display all object references in the program. If the SQL naming convention is used, the library name is stored in the OIR in one of two ways:

1. If the SQL name is fully qualified, the data base name is stored.
2. If the SQL name is not fully qualified, the authorization ID of the statement is stored.

If the system naming convention is used, the library name is stored in the OIR in one of two ways:

1. If the object name is fully qualified, the library name is stored.
2. If the object is not fully qualified, *LIBL is stored.

Running a Program with Embedded SQL

Running a host language program with embedded SQL statements, after the precompile and compile have been successfully done, is the same as running any host program. Type:

```
CALL pgm-name
```

on the system command line. For more information on running programs, see the *CL Programmer's Guide*

Blocking Considerations

SQL attempts to retrieve and insert records a block at a time when possible to improve performance. The user can control blocking, if desired, by using the SEQONLY parameter on the OVRDBF CL command prior to calling the application program that contains the SQL statements. For more information on the OVRDBF command, see the *CL Reference* manual.

---

[2] For more information about the DSPPGMREF CL command, see the *CL Reference* manual.

SQL automatically blocks records in the following cases:

- INSERT

  If an INSERT statement contains a subselect and if COMMIT(*NONE) has been specified, inserted records are blocked and not actually inserted into the target table until the block is full.

  If an INSERT statement contains a subselect, and COMMIT(*ALL) has not been specified, records for the subselect are retrieved in a block.

  Note: If an INSERT with a VALUES list is specified, SQL will not actually close the internal cursor used to perform the inserts until the program ends. If the same INSERT statement is run again, a full open is not necessary and the application runs much faster.

- OPEN

  If COMMIT(*NONE) has been specified, and the cursor is only used for FETCH statements, blocking is done when the records are retrieved. Note that if dynamic SQL statements exist in the program, SQL is unable to determine whether UPDATE or DELETE statements refer to the cursor, so blocking is not done.

OS/400 DDM Considerations

SQL does not support remote file access through OS/400 DDM (distributed data management).

Override Considerations

You can use overrides (specified by the OVRDBF command) to redirect a reference to a different table or view or change certain operational characteristics of the program. The following parameters are processed if an override is specified:

TOFILE
MBR
SEQONLY
LVLCHK
INHWRT
WAITRCD

All other override parameters are ignored. For more information on overrides, see the *Data Base Guide* and the *Data Management Guide*.

SQL Return Codes

A list of SQL return codes is provided in Appendix C.

Chapter 10. Using Interactive SQL

This chapter describes how to use interactive SQL to syntax check, validate, and run SQL statements. An overview, functional description, and tips on using interactive SQL are provided.

Overview

Interactive SQL allows you to enter SQL statements directly from the keyboard. Either a completion message or an error message is displayed after each statement is processed. In addition, status messages are normally displayed during long running statements. In the case of the SELECT statement, the result is also displayed, printed, or sent to a data base file.

There are three basic functions supplied by interactive SQL:

- The **statement entry** function allows you to type the interactive SQL statement and run it by pressing the Enter key.

  While in the statement entry function, you may:

  - Type in an interactive SQL statement and run it
  - Page through previous statements and messages
  - Call session services
  - Call list selection
  - Edit statements

- The **session services** function allows you to:
  - Change commitment control attributes
  - Change the values that are in effect when you run your interactive SQL statements.
  - Change the SELECT output device
  - Change the list of data bases (libraries)
  - Change the list type to select either all your system and SQL objects or only your SQL objects
  - Print the current session history
  - Remove all entries you made for the current session
  - Save the session in a source file
  - Change the data refresh operation when displaying data

- The **list selection** function allows you to select from lists of your authorized data bases, tables, views, or columns. The selections you make from the lists can be inserted into the SQL statement at the position located by the cursor.

Online help is available for all the displays in interactive SQL, as well as other normal system display operations.

If your previous SQL session ended abnormally, the first display that appears when you start a new SQL session is the Recover SQL Session display. The Recover SQL Session display allows you to continue with the old session or start a new session. If you do not recover the old session at this time, the old session is deleted.

You cannot run all of the SQL statements in the interactive SQL environment. For a list of valid interactive statements, see the "Supported SQL Statements" on page 10-8. However, you can use interactive SQL to check the syntax of all the SQL statements.

The following sections describe in more detail the functions available.

Terminology

There are two naming conventions that can be used on SQL/400: system (*SYS) and SQL (*SQL). The naming convention used affects the terms used on the displays and the method for qualifying file and table names. In the system naming convention, files are qualified by library name in the form: library/file. In the SQL naming convention, tables are qualified by the data base name in the form: database.table. The naming convention used is selected by a parameter on the Start SQL (STRSQL) command[1] on a command line. In the following discussion, the SQL naming convention is used.

The following lists shows the relationship between AS/400 object names and SQL/400 object names:

System Terms	SQL Terms
Library	Data base
Physical file	Table
Record	Row
Field	Column
Logical file	View

For further descriptions of the above listed terms, see the section "Relational Data Bases and Terminology" on page 1-2.

---

[1] For a description of the STRSQL command, see the "STRSQL Command" on page 10-15.

Getting Started

You can start using interactive SQL by entering STRSQL[1] on a command line. The Enter SQL Statements display then appears. This is the primary display from which you can enter SQL statements and, by using the function keys, use the session services (F13) and list select functions (F16 = Select data bases, F17 = Select tables, or F18 = Select columns).

```
                          Enter SQL Statements

Type SQL statement, press Enter.
===>
```

Press F24 to view function keys F16, F17, and F18.

```
F16=Select data bases  F17=Select tables  F18=Select columns  F24=More keys
          (libraries)           (files)           (fields)
```

Note: The names in parentheses appear in place of the ones above, if you select the system naming convention.

Audience

The intended users of interactive SQL are programmers and data base administrators with a knowledge of data base concepts and the SQL language.

Functional Description

Interactive SQL allows the programmer or data base administrator to quickly and easily define, update, delete, or look at data, mainly for testing, problem analysis, and/or data base maintenance. A programmer may, for example, insert some rows into a table, using interactive SQL and test their SQL statements before running them in an application program. A data base administrator may use interactive SQL to grant or revoke privileges, create or drop data bases, tables, or views, or select information from system catalog tables.

Statement Entry

The statement entry function is the function that you are in when first entering interactive SQL. You return to the statement entry function after processing each interactive SQL statement.

In the statement entry function, you type the entire SQL statement and submit it for processing by pressing the Enter key.

The statement may be one or many lines long. When the statement has been processed, the statement and the resulting text message are moved upward on the display. You then can enter another statement.

If the statement was recognized by the SQL parser but contains a syntax error, the statement that was just entered and the resulting text message (syntax error) is moved upward on the display, and a copy of the statement with the error appears in the input area with the cursor positioned at the beginning of the token that contains the syntax error.

You may page through previous statements, commands and/or messages—even those that have moved off of the top of the display. If you need more room to enter an SQL statement, you can page down the display.

Session Services

The session services function is made available by pressing F13 (Session services) on the Enter SQL Statements display, which accesses the Work with Session Services display. It is through this function that you can change the current values that are in effect for your interactive SQL session. You also can print, clear, or save the session to a source file. The following is a brief description of the options on the Work with Session Services display:

1. Change commitment control

   This function lets you set the level of commitment control that you want in effect for your session. The choices are *NONE, *CHG, and *ALL.

   Note: If locks on rows are currently being held for this unit of recovery and you attempt to change COMMIT from *ALL or *CHG to *NONE, a warning message is displayed.

   For more information on the COMMIT and ROLLBACK statements, see the *SQL/400 Reference* manual.

2. Change statement processing control

   This function lets you control how your SQL statement is processed. You can do one of the following:

   a. Check for correct syntax, make a validity check, and run your SQL statement (the Run option).
   b. Check for correct syntax and check that the object names used in the statement actually exist on your system (called a validity check), but not run the statement. This option does not allow the use of host variables.
   c. Check for correct syntax only. This option allows host variables.

   When you run the statement, a syntax check and validity check are performed automatically.

3. Change SELECT output device

   This function lets you control where the output from a successful SELECT statement is sent. You can either display, print, or save the results in a data base file. If you choose to print or save the results, you are prompted for printer or data base file information.

4. Change list of data bases (libraries)

   This function lets you control which data bases and libraries are used as a basis for building the data base list when F16 is pressed. You can select from *LIBL, *USRLIBL, *ALLUSR, *ALL, and *CURLIB, or specify a library name. The

name you specify may be used to supply a single data base or library. You are prompted for the name. For more information on the other five choices, see the *CL Reference* manual.

When you change the data base (library) option, it affects the presentation of the lists only and not the parser. In the system naming convention, the *LIBL list is used to resolve an unqualified file name, regardless of the setting of the list of libraries or of the STRSQL LIBOPT parameter. In the SQL naming convention, the user profile is used to resolve an unqualified table name. If there is a data base with the same name as your user profile, that data base name is used to resolve the unqualified table, regardless of the setting of the list of data bases or of the STRSQL LIBOPT parameter.

In either naming convention, selecting the file or table name from the list ensures that the name is properly qualified.

5. Change list type

   This function lets you control which types of objects are shown when you request a list. You may choose to see both system created objects and SQL created objects, or SQL created objects only.

6. Print current session

   This function lets you print the current session immediately and then continue working. You are prompted for printer information. All the SQL statements you entered and all the messages displayed are printed just as they appear on the Enter SQL Statements display.

7. Remove all entries from current session

   This function lets you remove all the SQL statements and messages from the Enter SQL Statements display and the session history. You are prompted to ensure that you really want to delete the information.

8. Save session in source file

   This function lets you save the session in a source file. You are prompted for the source file name. This function lets you embed the source file into a host language program by using the source entry utility (SEU).

9. Change data refresh option

   This option lets you control the data refresh operation when using the interactive SELECT statement to display data. You are prompted to specify whether the data is always refreshed or whether the data is refreshed on the first forward pass only of the retrieval operation.

List Selection Function

The list selection function is made available by pressing F16, F17, or F18 on the Enter SQL Statements display. After pressing the desired function key, you are presented with a list of authorized data bases, tables and views, or columns from which to choose. If you request a list of tables, but you have not previously selected a data base, you are asked to select a data base first. If you request a list of columns, but you have not previously selected a table and/or a data base, you are asked to select a data base and/or table first.

The list selection mode allows you to do the following:

- Select one or more items from the list, numerically specifying the order.
- Page through the list.
- Display the list again (all input fields are erased).
- Obtain help.
- Exit the list selection function without making any choices.

When returning from the list function, the selections you made are inserted at the position of the cursor on the Enter SQL Statements display. When using the table and view list to insert names into you statement, a maximum of 32 tables and/or views may be selected from the list. For more tips on using the list selection function, see "Using the List Selection Function" on page 10-11.

Note: **An important rule for using the list function:** Always select the list you are primarily interested in. For example, if you want a list of columns, but you believe that the column(s) you want are in a data base and/or table not currently selected, press F18 (Select columns). Then, from the column list, press the appropriate function key to change the table and/or data base. Do not go first to the data base or table list, in this case, because you do not want to have the table name or data base name inserted into your statement.

For more information on using the list selection function, see "Using the List Selection Function" on page 10-11.

Exit Interactive SQL

Function key F3 allows you to exit the interactive SQL environment and either:

1. Exit the session; the session history is not saved.
2. Save the session in a source file (this is the same function as option 8 on the Work with Session Services display).
3. Decide not to exit the session and return to the Enter SQL Statements display; the current session parameters remain in effect.

Note: If locks on rows are currently being held for this unit of recovery and you attempt to exit interactive SQL, a warning message is displayed.

Help

Help is available from every display in the interactive SQL session. You can obtain general help information about the whole display, or selective information about part of the displayed information by positioning the cursor in the area in question and pressing the Help key.

The Session and Its Functions

When you enter the start SQL command (STRSQL)[2] an interactive SQL session is created. The session consists of:

- Values of parameters you specified in the STRSQL command
- SQL statements you entered in the session along with their corresponding messages following each SQL statement
- Values of the parameters that you changed via the session services function, if applicable
- Lists selections you have made

Interactive SQL supplies a unique session-id. This allows multiple users, with the same sign-on ID, to use interactive SQL from more than one work station at the same time. Also, more than one interactive SQL session can be run from the same work station at the same time by the same user (sign-on ID).

Recovering a Failed SQL Session

If the previous SQL session ended abnormally, interactive SQL presents the Recover SQL Session display at the start of the next session (when the next STRSQL command is entered). From this display, you can either recover the old session by selecting option 1 or delete the old session and start a new session by selecting option 2. If you choose to recover the old session, the parameters you specified when you entered STRSQL are ignored and the parameters for the old session are used. If you choose to delete the old session and continue with the new session, the parameters you specified when you entered STRSQL are used.

Messages

Only first-level text for messages are shown on the Enter SQL Statements display. You can see the second-level text by positioning the cursor to the first-level text and pressing the Help key.

If an error is detected in a SQL statement, the statement is duplicated and the cursor is positioned to the place in the statement where the error was detected.

---

[2] For a description of the STRSQL command, see the "STRSQL Command" on page 10-15.

Supported SQL Statements

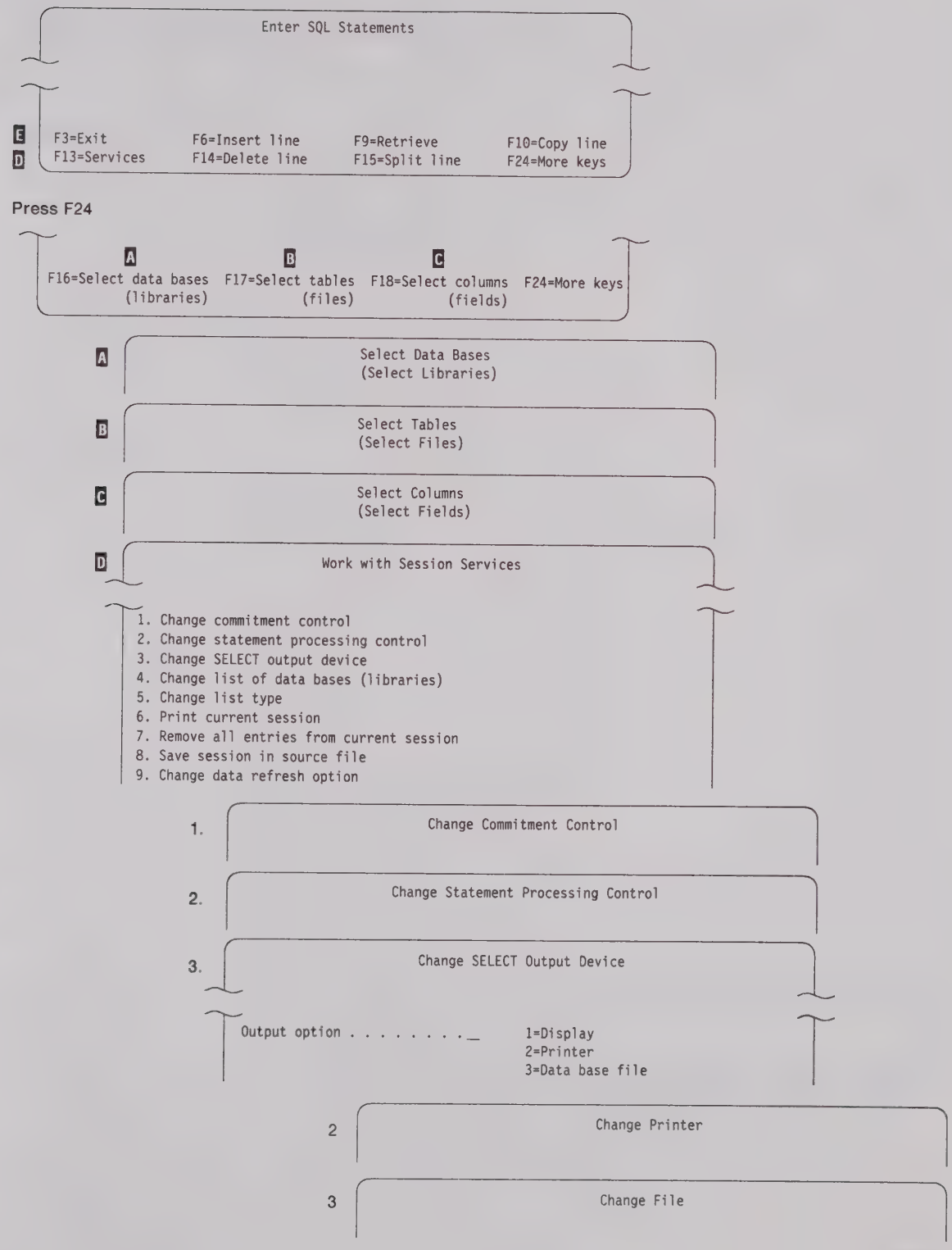
The following SQL statements can be run in the interactive SQL environment. All SQL statements can be syntax checked by interactive SQL. See the *SQL/400 Reference* manual for information about the statements.

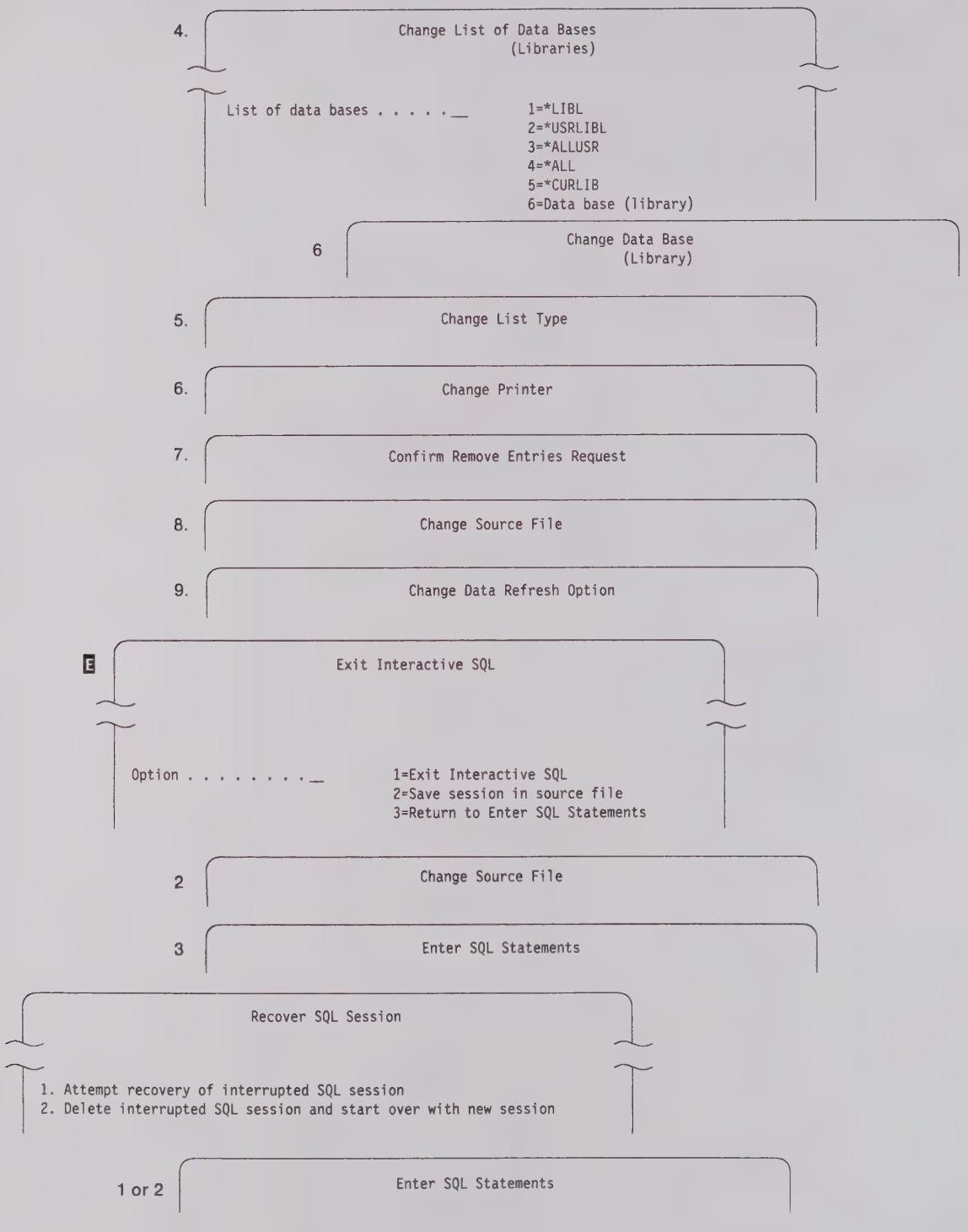
In the case where there is a distinction between the interactive form of the SELECT statement and other forms, the interactive form must be used.

COMMENT ON	GRANT
COMMIT	INSERT
CREATE DATABASE	LABEL ON
CREATE INDEX	LOCK TABLE
CREATE TABLE	REVOKE
CREATE VIEW	ROLLBACK
DELETE	SELECT
DROP	UPDATE

Interactive Session Display Flow Diagram

The following figure shows the overall flow of the displays used in an interactive SQL session. Remember, the only statement that returns data is the SELECT statement.


Enter SQL Statements
E
D
F3=Exit
F13=Services
F6=Insert line
F14=Delete line
F9=Retrieve
F15=Split line
F10=Copy line
F24=More keys
Press F24
A
B
C
F16=Select data bases (libraries)
F17=Select tables (files)
F18=Select columns (fields)
F24=More keys
A
Select Data Bases
(Select Libraries)
B
Select Tables
(Select Files)
C
Select Columns
(Select Fields)
D
Work with Session Services
1. Change commitment control
2. Change statement processing control
3. Change SELECT output device
4. Change list of data bases (libraries)
5. Change list type
6. Print current session
7. Remove all entries from current session
8. Save session in source file
9. Change data refresh option
1.
Change Commitment Control
2.
Change Statement Processing Control
3.
Change SELECT Output Device
Output option . . . . . _
1=Display
2=Printer
3=Data base file
2
Change Printer
3
Change File
RSLS751-7


4.
Change List of Data Bases
(Libraries)
List of data bases . . . . . __
1=*LIBL
2=*USRLIBL
3=*ALLUSR
4=*ALL
5=*CURLIB
6=Data base (library)
6
Change Data Base
(Library)
5.
Change List Type
6.
Change Printer
7.
Confirm Remove Entries Request
8.
Change Source File
9.
Change Data Refresh Option
E
Exit Interactive SQL
Option . . . . . __
1=Exit Interactive SQL
2=Save session in source file
3=Return to Enter SQL Statements
2
Change Source File
3
Enter SQL Statements
Recover SQL Session
1. Attempt recovery of interrupted SQL session
2. Delete interrupted SQL session and start over with new session
1 or 2
Enter SQL Statements
RSLS752-7

Tips on Using Interactive SQL

The following paragraphs contain tips and techniques for using interactive SQL.

Using the List Selection Function

You may request a list at any time while typing an SQL statement on the Enter SQL Statements display. The selections you make from the lists are inserted on the Enter SQL Statements display, starting where the cursor is located and in the order that you specify. You must type the key words of the statement, but the selected list information is added automatically.

The example on the following pages shows you how to use the list select function to build your SELECT statement.

Note: The following example lets you see how you might obtain the specific data you desire. The object names and selections are imaginary and probably do not exist on your system.

First Time Used in Session

First, assume you have *just entered* interactive SQL by typing STRSQL on the system Command Entry display, and you have made no list selections or entries. Also, assume you have selected *SQL for the naming convention.

At the Enter SQL Statements display:

1. Type SELECT on the first data entry line.
2. Type FROM on the second data entry line.
3. Leave the cursor positioned just after FROM.

```
                          Enter SQL Statements

Type SQL statement, press Enter.
===> SELECT
     FROM _
```

4. Press F17 (Select tables) to obtain a list of tables, because you want the table name to follow FROM.

   Note: Instead of a list of tables appearing as you expected, a list of data bases appears (the Select Data Bases display). This is because you have not selected a data base or data bases from which to work, since you just entered the SQL session.

5. Type a 1 in the *Seq* column across from YOURDB2 data base.

```
                          Select Data Bases

SQL statement . . . . . :
     SELECT
     FROM


Type sequence number(s)  (1-999) to select data base(s), press Enter.

Seq    Data Base      Type   Text
        YOURDB1       USER   Company benefits
 1      YOURDB2       USER   Employee personal data
        YOURDB3       USER   Job classifications/requirements
        YOURDB4       USER   Company insurances
```

6. Press the Enter key.

   The Select Tables display appears, showing the tables existing in the YOURDB2 data base.

7. Type a 1 in the *Seq* column across from PEOPLE table.

```
                          Select Tables

SQL statement . . . . . :
     SELECT
     FROM


Type sequence number(s) (1-999) to select table(s), press Enter.

Seq    Table/View    Data Base     Type   Text
        EMPLCO       YOURDB2       TABLE  Employee company data
 1      PEOPLE       YOURDB2       TABLE  Employee personal data
        EMPLEXP      YOURDB2       TABLE  Employee experience
        EMPLEVL      YOURDB2       TABLE  Employee evaluation reports
        EMPLBEN      YOURDB2       TABLE  Employee benefits record
        EMPLMED      YOURDB2       TABLE  Employee medical record
        EMPLINVST    YOURDB2       TABLE  Employee investments record
```

8. Press the Enter key.

   The Enter SQL Statements display appears again, with the table name, YOURDB2.PEOPLE, inserted after FROM.

   The table name is qualified by the data base name and both names appear in the proper syntax.

```
                        Enter SQL Statements

SQL statement . . . . . :
===> SELECT
     FROM YOURDB2.PEOPLE _
```

9. Position the cursor just after SELECT.
10. Press F18 (Select columns) to obtain a list of columns, because you want the column name to follow SELECT.

    The Select Columns display appears, showing the columns existing in the PEOPLE table.
11. Type a 1 in the *Seq* column across from the *NAME* column.
12. Type a 2 in the *Seq* column across from the *SOCSEC* column.

```
                          Select Columns

SQL statement . . . . . :
     SELECT _
     FROM YOURDB2.PEOPLE


Type sequence number(s) (1-999) to select table(s), press Enter.

Seq    Column      Table/view    Data Base    Text
 1     NAME        PEOPLE        YOURDB2
       EMPLNO      PEOPLE        YOURDB2      Employee ID no.
 2     SOCSEC      PEOPLE        YOURDB2
       COADDR      PEOPLE        YOURDB2      Company mail address
       STRADDR     PEOPLE        YOURDB2      Street address
       CITY/ST     PEOPLE        YOURDB2
       ZIP         PEOPLE        YOURDB2
       COPHONE     PEOPLE        YOURDB2      Employee internal telephone number.
       PERPHONE    PEOPLE        YOURDB2      Employee home telephone number.
       AGE/SEX     PEOPLE        YOURDB2
```

13. Press the Enter key.

    The Enter SQL Statements display appears again, with the NAME, SOCSEC appearing after SELECT.

```
                        Enter SQL Statements

SQL statement . . . . . :
===> SELECT NAME, SOCSEC
     FROM YOURDB2.PEOPLE
```

14. Press the Enter key.

The statement you created is now syntax checked, validity checked, and/or run, according to the statement processing options that you previously selected.

If you ran the statement and if no errors were encountered, the SQL Statement Entry display looks like this:

```
                          Enter SQL Statements

SQL statement . . . . . :
     SELECT NAME, SOCSEC
     FROM YOURDB2.PEOPLE
     SELECT statement run complete.
===>
```

After First Use in Session

Once you have used the list function, the values you selected remain in effect until you change them or until you change the list of libraries option on the Work with Session Services display.

Note: To change the lists, remember this rule:

> Always select the list you are primarily interested in. For example, if you want a list of columns, but you believe that the column(s) you want is in a data base and/or table not currently selected, press F18 (Select columns). Then from the column list display, press the appropriate function key to change the table and/or data base. Do not go first to the data base or table list, in this case, because you do not want to have the table name or data base name inserted into your statement.

Testing Your SQL Statements Using Interactive SQL

An important use for interactive SQL is to test your SQL statements before embedding them into your host program. When exiting interactive SQL, save your session into a source file. Then, use the system source entry utility (SEU) to copy the statement(s) into your program.

Entering DBCS Data

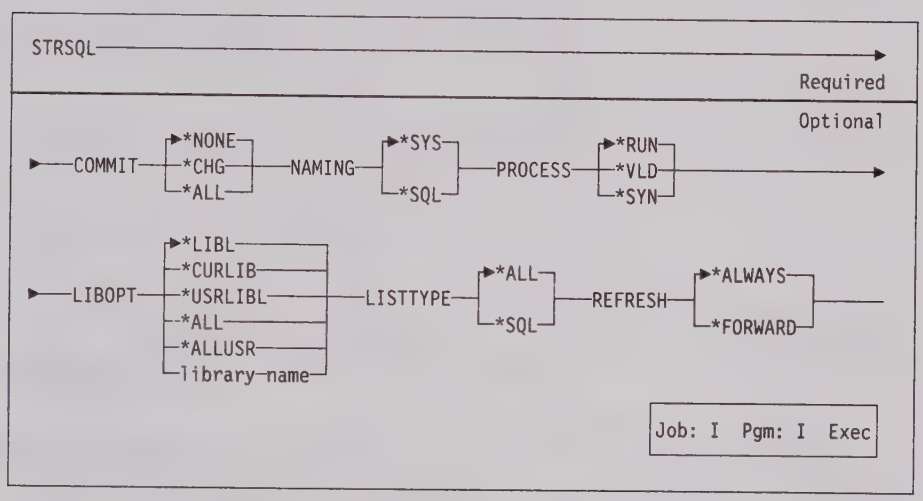
When you enter double-byte character set (DBCS) data on the Enter SQL Statements display, you must be aware of how the shift-out and shift-in characters are processed by interactive SQL. Each line of data must contain as many shift-out characters as shift-in characters. To assist processing a DBCS data string requiring more than one line for entry, interactive SQL removes the extra shift-out and shift-in characters. If the very last (farthest right) column on the line contains a shift-in character and the very first (left most) column of the next line contains a shift-out character, these shift-in, shift-out characters are removed by interactive SQL when the statement is processed. Also, when the two farthest right columns of a line contain a shift-in character, followed by a single-byte blank character, and the next line contains a shift-out character in the farthest left column, the shift-in, blank, shift-out sequence is removed when the statement is processed.

STRSQL Command

The STRSQL command starts interactive SQL, which immediately shows the Enter SQL Statements display. This display allows you to build, edit, enter, and run a SQL statement in an interactive environment. Any messages during the running of the program are shown on this display.

You can also specify many processing options by typing STRSQL on the command line and then pressing F4 (prompt). All of the parameters for the STRSQL command, except NAMING, can be set by using the Work with Session Services display in interactive SQL.

A syntax diagram of the command and a definition of the parameters follows:



COMMIT

Specifies whether the SQL statements are run under commitment control.

***NONE:** Specifies that commitment control is not used. COMMIT and ROLLBACK statements are not allowed. If SQL DDL are run, *NONE must be specified.

***CHG:** Specifies that only those rows updated, deleted, or inserted are locked until the transaction is committed or rolled back.

***ALL:** Specifies that every row selected, updated, deleted, and inserted are locked until the transaction is committed or rolled back.

NAMING

Specifies the naming convention used for naming objects in SQL statements.

***SYS:** Indicates that the system naming convention is used (library-name/file-name).

***SQL:** Indicates that the SQL naming convention is used (data-base-name.table-name).

PROCESS

Specifies what values are used to process the statements.

***RUN:** Specifies that the statements are syntax checked, data checked, and then run.

***VLD:** Specifies that the statements are syntax checked and data checked, but not run.

***SYN:** Specifies that the statements are syntax checked only.

LIBOPT

Specifies which data bases and libraries are used as a basis for building a data base list when F16 is pressed.

***LIBL:** Specifies that all the libraries in the user and system portions of the job's library list are shown.

***CURLIB:** Specifies that the current library is shown.

***USRLIBL:** Specifies that only the libraries in the user portion of the job's library list are shown.

***ALLUSR:** Specifies that all the nonsystem libraries, which include all user-defined libraries and the QGPL library, that are not in the job's library list are shown. Libraries whose names start with the letter Q, other that QGPL, are not included.

***ALL:** Specifies that all the libraries in the system, including QSYS, are shown.

library-name: Allows you to specify the name of a library.

LSTTYP

Specifies what types of objects are displayed with with list support (F16, F17, and F18).

***ALL:** Specifies that all objects are displayed.

***SQL:** Specifies that only SQL created objects are displayed.

REFRESH

Specifies when the displayed SELECT output data is refreshed.

***ALWAYS:** Specifies that the data is normally always refreshed.

***FORWARD:** Specifies that the data is refreshed only as the user scrolls forward to the end of the data for the first time. After that, it is not refreshed.

Example

```
STRSQL COMMIT(*CHG) NAMING(*SQL)
```

This command starts an interactive SQL session that locks only those rows that are updated, deleted, and inserted until the transaction is committed or rolled back. The SQL naming convention is used for this session.

Chapter 11. SQL Data Protection

This chapter describes the security plan for protecting SQL data from unauthorized users and the methods for ensuring data integrity.

SQL Security

All objects on the AS/400 system, including SQL objects, are managed by the system security function. SQL uses the GRANT and REVOKE statements to interface with the AS/400 system commands Grant Object Authority (GRTOBJAUT) and Revoke Object Authority (RVKOBJAUT). For more information on system security and the use of the GRTOBJAUT and RVKOBJAUT commands, see the *Security Concepts and Planning* manual.

The SQL GRANT and REVOKE statements only operate on tables and views. In some cases, it is necessary to use system GRTOBJAUT and RVKOBJAUT commands to authorize users to other objects, such as commands and programs.

The authority for SQL statements depends on whether the statement is static, dynamic, or being run interactively.

For static SQL statements, authority is checked against the user who is the owner of the program containing the SQL statement in addition to the user who is running the program. This is done when the precompiler calls the compiler, using USRPRF(*OWNER).

For dynamic SQL statements or for statements issued interactively, authority is based on the user who is running the program or is processing the statement.

Authorization ID

The authorization ID identifies a unique user and, in SQL/400, is a user profile object. Authorization IDs can be created using the system Create User Profile (CRTUSRPRF) command.

Public Authority

Public authority on specific tables and views is controlled by using the SQL GRANT and REVOKE statements. If *PUBLIC is specified with the SQL GRANT statement, authority to an object is granted to all users that have no private authority to that object.

For more information on the GRANT and REVOKE statements, see the *SQL/400 Reference* manual.

Views

A view can prevent unauthorized users from having access to sensitive data. The application program can access the data it needs in a table, without having access to sensitive or restricted data in the table (for example, employee salaries).

SQL Data Integrity

SQL data integrity protects data from being destroyed or changed by unauthorized persons, system operation or hardware failures (such as physical damage to a disk), programming errors, interruptions before a job is completed (such as a power failure), or interference from running applications at the same time (such as serialization problems). Data integrity is ensured by the following functions:

- Concurrency
- Atomic operations
- Journaling
- Commitment control
- Save/restore
- Damage tolerance
- Index recovery

For more information about each of these functions, see the *Data Base Guide* and the *Backup and Recovery Guide*.

Concurrency

Concurrency is the ability for multiple users to access and change data in the same table or view at the same time without risk of losing data integrity. This ability is automatically supplied by the AS/400 data base manager. Locks are implicitly acquired on tables and rows to protect concurrent users from changing data the same data at the same time.

In some cases, the program may acquire locks that prevent other statements in the same program from running. For example, a lock on a row currently held by one cursor will prevent another cursor in the same program (or in a DELETE or UPDATE statement not associated with the cursor) from acquiring a lock on the same row.

Deadlock detection is not provided; instead, default and user-specifiable lock-wait time-out values are supported. SQL creates tables, views, and indexes with the default record wait time (60 seconds) and the default file wait time (*IMMED). The user may change these values by using the Change Physical File (CHGPF) and Change Logical File (CHGLF) commands. (For more information on these commands, see the *CL Reference* manual.)

You may explicitly prevent other users from using a table by using the SQL LOCK TABLE statement, which is described in the *SQL/400 Reference* manual.

Atomic Operations

In general, all underlying data base data definition functions are designed to be atomic (either they will complete or they will appear to never have been started). This is true regardless of when or how the function was ended or interrupted (power failure, abnormal ending, job cancel, and so forth). Data definition statements are not affected by the COMMIT and ROLLBACK statements. However, because the underlying data base data definition functions are atomic, the data bases are never left in an unusable state.

The following SQL data definition statements are guaranteed to be atomic:

COMMENT ON	GRANT (See note.)
DROP TABLE	LABEL ON
DROP VIEW	REVOKE (See note.)
DROP INDEX	

Note: If multiple tables are specified for a GRANT or REVOKE statement, only the GRANT or REVOKE to one of the tables will be atomic. To make the GRANTs and REVOKEs to all the tables atomic, you must issue a GRANT or REVOKE against each table, individually.

The following data definition statements are not atomic because they involve more than one data base operation:

CREATE DATABASE
CREATE TABLE
CREATE VIEW
CREATE INDEX
DROP DATABASE

For example, a CREATE TABLE may be interrupted after the AS/400 physical file has been created, but before the member has been added. Therefore, in the case of create statements, if an operation ends abnormally, you may have to drop the object and then create it again. In the case of a DROP DATABASE statement, you may have to drop the data base again.

Journaling

The AS/400 journal support supplies an audit trail and forward and backward recovery. Forward recovery can be used to take an older version of a table and apply the changes logged on the journal to the table. Backward recovery can be used to remove changes logged on the journal from the table.

When a data base is created, a journal and journal receiver are created in the data base. The journal and journal receiver are not created on a user auxiliary storage pool (ASP). However, because placing journal receivers on ASPs can improve performance, the user who manages the journal may wish to create all future receivers on an ASP.

When a table is created, it is automatically journaled to the journal SQL created in the data base. After this point, it is the user's responsibility to use the journal functions to manage the journal, the journal receivers, and the journaling of tables to the journal. For example, if a table is moved into a data base, no automatic change to the journaling status occurs. If a table is restored, the normal journal rules apply. That is, if the table was journaled at the time of the save, it is journaled to the same journal at restore time. If the table was not journaled at the time of the save, it is not journaled at restore time.

A user can stop journaling on any table using the journal functions, but doing so prevents SQL from running under commitment control. SQL is still able to function in this case if the user has specified COMMIT(*NONE); however, this does not provide the same level of integrity that journaling and commitment control provide.

Commitment Control

The AS/400 commitment control provides a means of grouping UPDATE, INSERT, and DELETE operations that allows the processing of a group of data base changes as a single unit through the SQL COMMIT statement, or the removal of a group of data base changes on a single unit through the SQL ROLLBACK statement.

If the user requests COMMIT (*CHG) and COMMIT (*ALL) when the program was precompiled or when interactive SQL was started, then SQL sets up the commitment control environment by implicitly evoking the Start Commitment Control (STRCMTCTL) command. The specified COMMIT keyword value is used when SQL starts commitment control. NFYOBJ(*NONE) is specified when SQL starts commitment control. The user can issue STRCMTCTL before evoking SQL in order to specify different NFYOBJ or LCKLVL parameters.

In the following cases, a table or view may be locked exclusively in order to satisfy COMMIT(*ALL):

- When the user issues a STRCMTCTL and specifies LCKLVL (*CHG) prior to starting interactive SQL.
- If a subselect is processed that includes a join, group by, or union, or the processing of the query requires the use of a temporary result, an exclusive lock is acquired to protect the user from seeing uncommitted changes.

If the user specifies commitment control and the application does not complete for any reason or if it requests a ROLLBACK, all updates, inserts, and deletes made within the unit of recovery are backed out, even if a power failure occurs.

The journal created in the SQL data base is normally the journal used for logging all changes to SQL tables. The user may, however, use the system journal functions to journal SQL tables to a different journal. This is necessary if tables from multiple data bases need to be used in the same application unit of recovery. This is because AS/400 commitment control requires that all files under commitment control are journaled to the same journal.

The AS/400 system uses locks to keep other applications from accessing changed data before a unit of recovery completes. If COMMIT(*ALL) is specified, locks are also used to hold the content of a record that was read by an application for the duration of a unit of recovery to ensure it is not changed by another unit of recovery before application completion. This ensures that, if the same unit of recovery rereads a record, it gets the same result.

Commitment control handles up to 4096 distinct row changes in a unit of recovery. If COMMIT(*ALL) is specified, all rows read are also included in the 4096 limit. (If a row is changed or read more than once in a unit of recovery, it is only counted once toward the 4096 limit.) Holding a large number of locks adversely affects system performance and does not allow concurrent users to access rows locked in the unit of recovery until the end of the unit of recovery. It is, therefore, in the user's best interest to keep the number of rows processed in a unit of recovery small.

The HOLD value on COMMIT and ROLLBACK allows the user to keep the cursor open and start another unit of recovery without reissuing an OPEN.

If there are locked rows (records) pending from running a SQL precompiled program or an interactive SQL session, a COMMIT or ROLLBACK statement can be issued from the system Command Entry display. Otherwise, an implicit ROLLBACK operation occurs the job is ended.

Commitment control does not apply to data definition statements.

Save/Restore

The AS/400 save/restore functions are used to save tables, views, indexes, journals, journal receivers, and data bases on disk (save file) or to some external media (tape or diskette). The saved versions may be restored onto the system at some later time. The save/restore function allows an entire data base, selected objects, or only objects changed since a given date/time to be saved. All information needed to restore an object to its previous state is saved. This function can be used to recover from damage to individual tables by reloading the data base with a previous version of the damaged table or the entire data base.

Damage Tolerance

The AS/400 data base functions have a certain amount of tolerance to damage caused by disk errors or system errors.

A DROP operation always succeeds, regardless of the damage. This assures that should damage occur, at least the table, view, or index can be deleted and restored or recreated.

In the event that a disk error has damaged a small portion of the rows in a table, the AS/400 data base manager allows the user to read rows still accessible.

Index Recovery

The AS/400 data base manager supplies several functions to deal with index recovery. All indexes on the system have a maintenance option that specifies when an index is maintained. SQL indexes are created with an attribute of *IMMED maintenance.

In the event of a power failure or abnormal system failure, indexes that are in the process of change may need to be rebuilt to make sure they agree with the actual data. All indexes on the system have a recovery option that specifies when an index should be rebuilt if necessary. All SQL indexes with an attribute of UNIQUE are created with a recovery attribute of *IPL (this means that these indexes are rebuilt before the OS/400 has been started). All other SQL indexes are created with the *AFTIPL recovery option (this means that after the operating system has been started, indexes are asynchronously rebuilt). During an IPL, the operator can see a display showing indexes needing to be rebuilt and their recovery option. The operator may override the recovery options.

The AS/400 data base manager supplies an index journaling function that makes it unnecessary to rebuild an entire index due to a power or system failure. If the index is journaled, the system data base support automatically makes sure the index is in synchronization with the data in the tables without having to rebuild it from scratch. SQL indexes are *not* journaled automatically. The user can, however, use the Start

Journal Access Path (STRJRNAP) CL command[1] to journal any index created by SQL.

The save/restore function allows the user to save indexes when a table is saved by using ACCPTH (*YES) on the Save Object (SAVOBJ) or Save Library (SAVLIB) CL commands. In the event of a restore when the indexes have also been saved, there is no need to rebuild the indexes. Any indexes not previously saved and restored are automatically and asynchronously rebuilt by the data base.

---

[1] For more information on control language commands, see the *CL Reference* manual.

Chapter 12. Testing SQL Statements in Application Programs

This chapter describes how to establish a test environment for and how to debug SQL statements in an application program.

Establishing a Test Environment

Some things you need to test your program are:

- **Authorization.** You need to be authorized to create tables and views, access SQL data, and create and run programs.
- **A test data structure.** If your program updates, inserts, or deletes data from tables and views, *you should use test data* to verify the running of the program. If your program only retrieves data from tables and views, you might consider using production-level data when testing your program. It is recommended, however, that you use the Start Debug (STRDBG) command with UPDPROD(*NO) to assure that the production level data does not accidentally get changed. See the chapter on testing in the *CL Programmer's Guide* for more information on debugging.
- **Test input data.** The input data your program uses during testing should be valid data that represents as many possible input conditions as you can think of. You cannot be sure that your output data is valid unless you use valid input data.

  If your program verifies that input data is valid, include both valid and invalid data to verify that the valid data is processed and the invalid data is detected.

  You might have to refresh the data for subsequent tests.

To test the program thoroughly, test as many of the paths through the program as possible. For example:

- Use input data that forces the program to run each of its branches.
- Check the results. For example, if the program updates a row, select the row to see if it was updated correctly.
- Be sure to test the program error routines. Again, use input data that forces the program to encounter as many of the anticipated error conditions as possible.
- Test the editing and validation routines your program uses. Give the program as many different combinations of input data as possible to verify that it correctly edits or validates that data.

Designing a Test Data Structure

To test an application that accesses SQL data, you may have to create test tables and views:

- **Test views of existing tables.** If your application does not change data and the data exists in one or more production-level tables, you might consider using a view of the existing tables. It is also recommended that you use STRDBG command with UPDPROD(*NO) to assure that the production level data does not accidentally get changed. See the chapter on testing in the *CL Programmer's Guide* for more information on debugging.

- **Test tables.** When your application creates, changes, or deletes data, you will probably want to test the application by using tables that contain test data. See Chapter 2 for a description of how to create tables and views.

  Alternatively, you may want to use the CRTDUPOBJ CL command to create a duplicate test table, view, or index. See the *CL Reference* manual for more information on using the CRTDUPOBJ command.

Authorization

Before you can create a table, you must be authorized to create tables and to use the data base in which the table is to reside. In addition, you must have authority to create and run the programs you want to test.

If you intend to use existing tables and views (either directly or as the basis for a view), you must be authorized to access those tables and views.

If you want to create a view, you must be authorized to create views and must have authorization to each table and view on which the view is based. For more information on specific authorities required for any specific SQL statement, see the *SQL/400 Reference* manual.

Debugging Your Program

Debugging your program with SQL statements is much the same as debugging your program without SQL statements. However, when running your program with SQL statements in the STRDBG environment, SQL puts a message in the job log about how the SQL statements ran. This message is an indication of the SQLCODE for the SQL statement. If the statement ran successfully, the SQLCODE value is zero, and a completion message is issued. A negative SQLCODE results in a diagnostic message. A positive SQLCODE results in an informational message.

The message is a 4-digit code prefixed by SQL. For example, an SQLCODE of −204 results in a message of SQL0204.

References to high-level language statement numbers in debug must be taken from the compile listing.

Chapter 13. Guidelines and Techniques for Using SQL

This chapter provides guidelines and techniques for using SQL statements in application programs. The chapter consists of two sections: The first section, "Guidelines for Using SQL Statements," provides guidelines for designing a program that uses SQL and system resources more efficiently. The second section, "Techniques for Solving Some Common Data Base Problems" on page 13-7, suggests techniques for using SQL statements in an application program and solving some common data base problems.

Guidelines for Using SQL Statements

This section provides some specific considerations and guidelines to help you fine-tune the SQL statements in an application program. As a general rule, you can ignore most of these guidelines and get correct results when accessing SQL data. These guidelines help you design a program that minimizes its use of SQL and system resources and that minimizes the time needed to access SQL data from a very large table.

The SQL language is a high-level language with much flexibility. Because of this, you can sometimes write a SELECT statement several different ways to retrieve the same data. However, the performance of different forms of a SELECT statement can vary greatly. In this section are several examples of alternative SQL statements. The recommendations given with each example are based on the relative performance of the example.

Some of the suggestions are easy to carry out without compromising any of the ease-of-use functions of SQL. Others, however, are complex and difficult to use. To determine if you need to make the effort to fine-tune the performance of your SQL statements, consider the following guidelines:

- If you are accessing a table of 10,000 rows or more, you should start thinking about the performance implications of your SQL statements.
- If you are accessing a table of 100,000 rows or more, you should seriously consider the performance implications of your SQL statements.
- If the SQL statement you issue requires the ordering of 1000 rows or more, you should consider trying to improve the performance of the ORDER BY.
- If you are accessing more than one table (for example, a join), you should consider trying to improve the SELECT statement performance.

You can improve performance by effectively using an SQL index and by effectively selecting data from two or more tables, described in the following paragraphs.

Effectively Using an SQL Index

SQL provides two basic means for accessing tables: a table scan (sequential) and an index-based (direct) retrieval. Index-based retrieval is usually more efficient than table scan. However, when a very large percentage of pages are retrieved, table scan is more efficient than index-based retrieval.

If SQL cannot use an index to access the data in a table, it will have to read all the data in the table. Very large tables present a special performance problem: the high cost of retrieving all the data in the table. The following suggestions help you to design code that allows SQL to take advantage of available indexes.

1. **Avoid numeric conversions.**

   When a column value and a host variable (or literal value) are being compared, try to specify the same data types. SQL does not use an index for the named column if the host variable or literal value has a greater precision than the precision of the column. If the two items being compared have different data types, SQL will have to convert one or the other of the values, which may result in inaccuracies (because of limited machine precision). For example, EDUCLVL is a halfword integer value (SMALLINT). Specify:

   ```
   ... WHERE EDUCLVL < 11 AND
             EDUCLVL >= 2
   ```

   instead of

   ```
   ... WHERE EDUCLVL < 1.1E1 AND
             EDUCLVL > 1.3
   ```

2. **Avoid character string padding.**

   Try to use the same data length when comparing a fixed-length character string column value to a host variable or literal value. SQL does not use an index if the literal value or host variable is longer than the column length. For example, EMPNO is CHAR(6) and DEPTNO is CHAR(3). Specify:

   ```
   ... WHERE EMPNO > '000300' AND
             DEPTNO < 'E20'
   ```

   instead of

   ```
   ... WHERE EMPNO > '000300 ' AND
             DEPTNO < 'E20 '
   ```

3. **Avoid the use of LIKE patterns beginning with % or _.**

   The percent sign (%), and the underscore (_), when used in the pattern of a LIKE predicate, specify a character string that is similar to the column value of rows you want to select. When used to denote characters in the middle or at the end of a character string, as in

   ```
   ... WHERE LASTNAME LIKE 'J%SON%'
   ```

   they produce efficient results. However, when used at the beginning of a character string, as in

   ```
   ... WHERE LASTNAME LIKE '%SON'
   ```

   they may prevent SQL from using any indexes that might be defined on the *LASTNAME* column to limit the number of rows scanned. You should therefore avoid using these symbols at the beginning of character strings, especially if you are accessing a particularly large table.

4. **Be aware that SQL does not use an index in the following instances:**

   - For a column it expects to be updated. For example, your program might include

     ```
     EXEC SQL
       DECLARE DEPTEMP CURSOR FOR
         SELECT EMPNO, LASTNAME, DEPTNO
           FROM USER1.TEMPL
           WHERE (DEPTNO = 'D11' OR
                  DEPTNO = 'D21') AND
                 EMPNO >= '000190'
           FOR UPDATE OF EMPNO, DEPTNO
     END-EXEC.
     ```

     even if you do not intend to update the employee's serial number. In this example, SQL cannot use either of the indexes available for the USER1.TEMPL table.

     SQL can operate more efficiently if the FOR UPDATE OF column list only names the column you intend to update: *DEPTNO*. Therefore, do not specify a column in the FOR UPDATE OF column list unless you intend to update the column.

   - For a column being compared with another column from the same row. For example:

     ```
     EXEC SQL
       DECLARE DEPTDATA CURSOR FOR
         SELECT DEPTNO, DEPTNAME
           FROM USER1.TDEPT
           WHERE DEPTNO = ADMRDEPT
     END-EXEC.
     ```

     Even though there is an index for *DEPTNO* and another index for *ADMRDEPT*, SQL will not use either index. The index has no added benefit because every row of the table needs to be looked at.

5. **An application program can do retrievals based on some key value in either of two ways:**

   a. An embedded SELECT statement can be used with a host variable, or

   b. A cursor could be declared using a host variable and a sequence of OPEN, FETCH, and CLOSE statements can be issued.

For example, suppose you want to FETCH a row from a table based on some value input from a display station:

```
EXEC SQL
  DECLARE EMPCSR CURSOR FOR
    SELECT EMPNO, LASTNAME, DEPTNO
      FROM USER1.TEMPL
      WHERE EMPNO = :EMPVAR
END-EXEC.

(Input an employee number from the display station.)

EXEC SQL
  OPEN EMPCSR
END-EXEC.

EXEC SQL
  FETCH EMPCSR
    INTO :EMPVAR, :LASTVAR, :DEPTVAR
END-EXEC.

EXEC SQL
  CLOSE EMPCSR
END-EXEC.

(Get the next employee number from the display station
 and repeat the OPEN, FETCH, and CLOSE.)
```

If the WHERE condition can be completely satisfied by doing key retrievals using an existing index, SQL will not actually close the cursor when the CLOSE statement is issued. The OPEN statement must still be issued, before the next FETCH, but since the full open is not done, the application runs much faster. The cursor is completely closed when the program ends.

Alternatively, the following embedded SELECT statement could be specified:

```
(Input an employee number from the display station.)

EXEC SQL
  SELECT EMPNO, LASTNAME, DEPTNO
    INTO :EMPVAR, :LASTVAR, :DEPTVAR
    FROM USER1.TEMPL
    WHERE EMPNO = :EMPVAR
END-EXEC.

(Get the next employee number from the display station
 and repeat the SELECT.)
```

If the WHERE condition can be completely satisfied by doing key retrievals using an existing index, SQL will leave open the internal cursor used to select the records until the program ends. If the SELECT statement is run again, a full open is not necessary and the application runs much faster.

Improving Performance When Selecting Data from Two or More Tables

If the SELECT statement you are considering accesses two or more tables, all the recommendations suggested in the previous section apply. The following suggestion is directed specifically to SELECT statements that access several tables.

You might want to provide redundant information when joining several tables. If you give SQL extra information to work with when requesting a join, it can better determine the best way to do the join. The additional information might seem redundant, but it is helpful to SQL. For example, instead of coding:

```
EXEC SQL
  DECLARE DEPTDATA CURSOR FOR
    SELECT USER1.TEMPL.DEPTNO, DEPTNAME, EMPNO, LASTNAME
      FROM USER1.TEMPL, USER1.TPROJ
      WHERE USER1.TEMPL.DEPTNO = USER1.TPROJ.DEPTNO AND
            USER1.TPROJ.DEPTNO = :DEPTNUM
END-EXEC.
 ...
```

provide SQL with a little more data in the WHERE clause:

```
EXEC SQL
  DECLARE DEPTDATA CURSOR FOR
    SELECT USER1.TEMPL.DEPTNO, DEPTNAME, EMPNO, LASTNAME
      FROM USER1.TEMPL, USER1.TPROJ
      WHERE USER1.TEMPL.DEPTNO = USER1.TPROJ.DEPTNO AND
            USER1.TPROJ.DEPTNO = :DEPTNUM AND
            USER1.TEMPL.DEPTNO = :DEPTNUM
END-EXEC.
 ...
```

SQL may be able to make a more informed decision about the best way to process the SELECT statement when given this redundant information.

For joins involving more than two tables, providing redundant information might become tedious. To improve performance and minimize the amount of redundant information, code a predicate that refers to an indexed column.

In the above example, assume that *DEPTNO* column of USER1.TEMPL is indexed but *DEPTNO* column of USER1.TPROJ is not. To improve the statement performance (without providing redundant information), you can code:

```
EXEC SQL
  DECLARE DEPTDATA CURSOR FOR
    SELECT USER1.TEMPL.DEPTNO, DEPTNAME, EMPNO, LASTNAME
      FROM USER1.TEMPL, USER1.TPROJ
      WHERE USER1.TEMPL.DEPTNO = USER1.TPROJ.DEPTNO AND
            USER1.TEMPL.DEPTNO = :DEPTNUM
END-EXEC.
 ...
```

Improving Performance by Reducing the Number of Opens

The number of opens in a program can significantly affect performance. An open occurs on:

- An OPEN statement
- Embedded SELECT statement
- An INSERT statement with a VALUES clause
- An UPDATE statement with a WHERE condition
- An UPDATE statement with a WHERE CURRENT OF cursor and SET clauses that refer to operators or functions
- A DELETE statement with a WHERE condition

An INSERT statement with a subselect requires two opens.

To minimize the number of opens, SQL/400 leaves a cursor open in the following cases and reuses the cursor if the statement is run again:

- An INSERT statement with a VALUES list
- An embedded SELECT statement when the WHERE clause can be completely satisfied by doing key retrievals from an existing index and when GROUP BY, HAVING, or column functions have not been specified.
- OPEN statement when the WHERE clause in the associated DECLARE CURSOR can be completely satisfied by doing key retrievals from an existing index and when GROUP BY, HAVING, or column functions have not been specified.

SQL/400 only reuses cursors opened by the same statement number. An identical statement coded later in the program does not reuse a cursor from any other statement. If the identical statement must be run in the program many times, code it once in a procedure and call the procedure to run the function.

Improving Performance when Paging Interactively Displayed Data

When interactively displaying data, the results of a SELECT statement are copied to a temporary file as you page forward through the file. Other users sharing the data base file(s) can make changes to the records while you are displaying the SELECT statement results. If you page backward to records that have already been displayed, the records shown are those in the temporary file instead of those in the updated data base file.

In large files, paging performance is usually degraded because of the updating or **refresh** operation, which dynamically retrieves the latest data directly from the data base file(s). Paging performance can be improved by specifying the REFRESH(*FORWARD) parameter of the STRSQL command. This command option specifies that the SELECT statement results will be copied from the data base file(s) to a temporary file on the first forward pass only. Afterward, paging backward or forward is done in the temporary file. However, in this case, any changes made by other users will not be made to the temporary file.

The refresh option can be changed on the Session Services display.

Techniques for Solving Some Common Data Base Problems

This section provides techniques to help you do the following tasks:

- Paging through retrieved data
- Keeping a copy of the data
- Retrieving the data a second time
- Establishing position at the end of a table
- Adding data to the end of a table
- Updating data as it is retrieved from the data base
- Updating data previously retrieved
- Changing the table definition

Paging through Retrieved Data

When a program retrieves data from the data base, the FETCH statement allows it to page forward through the data. SQL has no statement equivalent to a backward FETCH. That leaves you with two programming options:

1. Keep a copy of the data that has been fetched and page through it by some programming technique.
2. Use SQL to retrieve the data again, typically by a second SELECT statement.

Both options are discussed in more detail in the following sections.

Keeping a Copy of the Data

One effect of this approach is that by paging backward you always see exactly the same data that was fetched, even if the data in the data base has changed in the meantime. That may be an advantage if you need to see a consistent set of data. However, it has the disadvantage of not allowing you to see updates made by others as soon as they are committed to the data base.

The locks you hold on the data may prevent others from updating it, which you may or may not want. If COMMIT(*ALL) is specified, the locks are not released after the data is fetched. Hence, if you want others to be able to update the data you have fetched, and you do not need to see those updates, commit your work after fetching the data.

Retrieving Data a Second Time

To retrieve the data a second time, the technique depends on the order in which you want to see the data again: either from the beginning or from the middle of the results table.

Retrieving from the Beginning

To retrieve the data again from the beginning, merely close the active cursor and reopen it. That positions the cursor at the beginning of the results table. But, unless the program holds an exclusive lock on the table locks on all the data, others may have changed it, and what was the first row of the results table is no longer the first row.

Retrieving from the Middle

To retrieve data a second time from somewhere in the middle of the results table, run a second SELECT statement and declare a second cursor on it. For example:

```
EXEC SQL
  DECLARE A CURSOR FOR
    SELECT * FROM DEPARTMENT
      WHERE LOCATION = 'MINNESOTA'
      ORDER BY DEPTNO
END-EXEC.
```

Suppose also that you now want to return to the rows that start with DEPTNO = 'M95', and fetch sequentially from that point. Declare a second cursor:

```
EXEC SQL
  DECLARE B CURSOR FOR
    SELECT * FROM DEPARTMENT
      WHERE LOCATION = 'MINNESOTA'
      AND DEPTNO >= 'M95'
      ORDER BY DEPTNO
END-EXEC.
```

That statement positions the cursor where you want it.

Again, unless the table was locked, rows may have been inserted or deleted by other users. If so, the row with DEPTNO = 'M95' may no longer exist. Or there may now be 20 rows with DEPTNO between M95 and M99, where before there were only 16.

The Order of Rows in the Second Results Table

The rows of the second results table may not appear in the same order. SQL does not consider the order of rows as significant unless the SELECT statement uses ORDER BY. Hence, if there are several rows with the same DEPTNO value, the second SELECT statement may retrieve them in a different order from the first. The rows are guaranteed to be in order by department number only if ORDER BY DEPTNO is specified.

The difference in ordering could occur even if you were to run the same SQL statement, with the same host variables, a second time. For example, indexes could be created or dropped that could affect the access plan.

The ordering is more likely to change if the second SELECT has a predicate that the first did not. SQL may choose to use an index on the new predicate. For example, SQL may choose an index on LOCATION for the first statement in our example, and an index on DEPTNO for the second. Because rows are fetched in order by the index key, the second order need not be the same as the first.

Again, executing PREPARE for two similar SELECT statements can produce a different ordering of rows even if no indexes are created or dropped. In the example, if there are many different values of LOCATION, SQL could choose an index on LOCATION for both statements. Yet, changing the value of DEPTNO in the second statement

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'MINNESOTA'
  AND DEPTNO >='Z98'
  ORDER BY DEPTNO
```

could cause SQL to choose an index on DEPTNO. Because of the subtle relationships between the form of an SQL statement and the values in it, never assume that two different SQL statements return rows in the same order, unless the order is uniquely determined by an ORDER BY clause.

Retrieving in Reverse Order

If there is only one row for each value of DEPTNO, then the following statement specifies a unique ordering of rows:

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'MINNESOTA'
  ORDER BY DEPTNO
```

To retrieve the same rows in reverse order, it is merely necessary to specify that the order is descending, as in this statement:

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'MINNESOTA'
  ORDER BY DEPTNO DESC
```

A cursor on the second statement would retrieve rows in exactly the opposite order from a cursor on the first statement. But that is guaranteed *only* if the first statement specifies a *unique* ordering.

For retrieving rows in reverse order, it might be useful to have two indexes on the *DEPTNO* column, one in ascending order and one in descending order.

Establishing Position at the End of a Table

The end of the table is defined as a result of an SQL SELECT statement like the one that follows:

```
SELECT * FROM DEPARTMENT
  ORDER BY DEPTNO DESC
```

Adding Data to the End of a Table

The order in which rows are returned to your program depends on the ORDER BY clause in the SQL statement. To get the effect of adding data to the end of a table, include a sequence number column in the table definition. Then, when you retrieve data from the table, use an ORDER BY clause naming that column.

Updating Data as It Is Retrieved from the Data Base

You can update rows of data as you retrieve them. On the SELECT statement, use FOR UPDATE OF, followed by a list of columns that may be updated. Then use the cursor-controlled UPDATE statement. The WHERE CURRENT OF clause names the cursor that points to the row you want to update. If neither a FOR UPDATE OF nor an ORDER BY clause is specified, all columns may be updated.

Restrictions

If the FOR UPDATE OF clause is specified, you cannot use FOR UPDATE OF with a SELECT statement that includes any of these elements:

- A column function (AVG, MIN, MAX, SUM, or COUNT)
- The keywords GROUP BY, UNION, or DISTINCT
- A column that is defined with an expression (for example, SALARY + COMMISSION)
- More than one table, as in a join

If a FOR UPDATE OF clause is specified, you cannot update columns that were not named in the FOR UPDATE OF clause. But you may name columns in the FOR UPDATE OF clause that are not in the SELECT list, as in this example:

```
SELECT A, B, C FROM TABLE
  FOR UPDATE OF A,E
```

Do not name more columns than you need in the FOR UPDATE OF clause; indexes on those columns are not used when you access the table.

Updating Data Previously Retrieved

You can page backward and update data that had been previously retrieved by doing one of two things:

1. If you have a second cursor on the data to be updated, and if the SELECT statement uses none of the above restricted elements, you can use a cursor-controlled UPDATE statement. Name the second cursor in the WHERE CURRENT OF clause.

   Note: This technique does not work for COMMIT (*ALL).

2. In other cases, use UPDATE with a WHERE clause that names all the values in the row or specifies a unique key of the table. You can code one statement, using host variables in the WHERE clause, and run the same statement many times with different values of the variables.

Changing the Table Definition

It is not possible to add, rearrange, or delete columns in a table without dropping and creating the entire table. However, you can dynamically create a view of the table, which includes only the columns you want, in the order you want.

Appendix A. SQL Reserved Words

This appendix contains words reserved by SQL/400. The reserved words may be used as delimited identifiers by enclosing them between quotation marks. All identifiers beginning with SQL are reserved.

ALL	IMMEDIATE
AND	IN
ANY	INDEX
AS	INSERT
BETWEEN	INTO
BY	IS
COLUMN	LIKE
COUNT	NOT
CURRENT	NULL
CURSOR	OF
DATABASE	ON
DELETE	OR
DESCRIPTOR	ORDER
DISTINCT	SELECT
DROP	SET
END-EXEC (COBOL)	TABLE
EXECUTE	TO
EXISTS	UNION
FOR	UPDATE
FROM	USER
GO	USING
GOTO	VALUES
GRANT	VIEW
GROUP	WHERE
HAVING	WITH

Note: Future releases of SQL may include additional reserved words. To avoid conflicts with future reserved words, you can use delimited identifiers in your application programs.

Appendix B. SQL Sample Tables

This appendix contains the sample tables referred to and used in the body of this manual. Along with the tables are the SQL statements for creating the tables and for inserting information into the tables. For detailed information on creating tables, see "Creating and Using a Table" on page 2-1. The tables are:

- Department table (TDEPT)
- Employee table (TEMPL)
- Project table (TPROJ)
- Employee project account table (TEMPRACT)

Creating the Tables

The SQL statements for creating the sample tables are shown in this section.

Creating the Department Table (TDEPT)

The sample department table describes each department in the company and specifies the department manager and the next higher department of authority.

This table can be created with the following interactive SQL statement:

```
CREATE TABLE USER1.TDEPT
    (DEPTNO     CHAR(3)     NOT NULL WITH DEFAULT,
     DEPTNAME   CHAR(36)    NOT NULL WITH DEFAULT,
     MGRNO      CHAR(6)     NOT NULL WITH DEFAULT,
     ADMRDEPT   CHAR(3)     NOT NULL WITH DEFAULT)
```

Creating the Employee Table (TEMPL)

The sample employee table describes the employees of the company.

This table can be created with the following interactive SQL statement:

```
CREATE TABLE USER1.TEMPL
    (EMPNO      CHAR(6)       NOT NULL WITH DEFAULT,
     FIRSTNME   CHAR(12)      NOT NULL WITH DEFAULT,
     MIDINIT    CHAR(1)       NOT NULL WITH DEFAULT,
     LASTNAME   CHAR(15)      NOT NULL WITH DEFAULT,
     DEPTNO     CHAR(3)       NOT NULL WITH DEFAULT,
     PHONENO    CHAR(4)       NOT NULL WITH DEFAULT,
     HIREDATE   CHAR(6)       NOT NULL WITH DEFAULT,
     JOBCODE    DECIMAL(3)    NOT NULL WITH DEFAULT,
     EDUCLVL    SMALLINT      NOT NULL WITH DEFAULT,
     SEX        CHAR(1)       NOT NULL WITH DEFAULT,
     BRTHDATE   CHAR(6)       NOT NULL WITH DEFAULT,
     SALARY     DECIMAL(8,2)  NOT NULL WITH DEFAULT)
```

Creating the Project Table (TPROJ)

The sample project table describes each programming project currently active in the company and specifies the department responsible for the project, the department manager, staffing information, and dates marking the duration of the project.

This table can be created with the following interactive SQL statement:

```
CREATE TABLE USER1.TPROJ
    (PROJNO     CHAR(6)       NOT NULL WITH DEFAULT,
     PRNAME     CHAR(36)      NOT NULL WITH DEFAULT,
     DEPTNO     CHAR(3)       NOT NULL WITH DEFAULT,
     DEPTMGR    CHAR(6)       NOT NULL WITH DEFAULT,
     PRSTAFF    DECIMAL(5,2)  NOT NULL WITH DEFAULT,
     PRSTDATE   CHAR(6)       NOT NULL WITH DEFAULT,
     PRENDATE   CHAR(6)       NOT NULL WITH DEFAULT,
     MAJPROJ    CHAR(6)       NOT NULL WITH DEFAULT)
```

Creating the Employee Project Account Table (TEMPRACT)

The sample employee project account table describes each programming project currently active in the company and identifies each employee involved in the project, the project number, the account number, the start and end dates of the employee's involvement, and the estimated number of hours the employee will charge to the project.

This table can be created with the following interactive SQL statement:

```
CREATE TABLE USER1.TEMPRACT
    (EMPNO      CHAR(6)       NOT NULL WITH DEFAULT,
     PROJNO     CHAR(6)       NOT NULL WITH DEFAULT,
     ACTNO      SMALLINT      NOT NULL WITH DEFAULT,
     STARTDATE  CHAR(6)       NOT NULL WITH DEFAULT,
     ENDDATE    CHAR(6)       NOT NULL WITH DEFAULT,
     EMPTIME    DECIMAL(5,2)  NOT NULL WITH DEFAULT)
```

Inserting Information into the Tables

Information (data values) is inserted into the tables by using the SQL INSERT statement (see "The INSERT Statement" on page 3-1).

Inserting Information into the TDEPT Table

The INSERT statement for inserting information into the department table is, in part, as follows:

```
INSERT INTO USER1.TDEPT
       (DEPTNO,
        DEPTNAME,
        MGRNO,
        ADMRDEPT)
  VALUES
       ('A00',
        'COMPUTER SERVICE DIV.',
        '000010',
        '      ')

...and so on for the rest of the information for the table.
```

Inserting Information into the TEMPL Table

The INSERT statement for inserting information into the employee table is, in part, as follows:

```
INSERT INTO USER1.TEMPL
       (EMPNO,
        FIRSTNME,
        MIDINIT,
        LASTNAME,
        DEPTNO,
        PHONENO,
        HIREDATE,
        JOBCODE,
        EDUCLVL,
        SEX,
        BRTHDATE,
        SALARY)
  VALUES
       ('000010',
        'CHRISTINE',
        'I',
        'HAAS',
        'A00',
        '3978',
        '750101',
         66,
         18,
        'F',
        '330814',
         52750)

...and so on for the rest of the information for the table.
```

Inserting Information into the TPROJ Table

The INSERT statement for inserting information into the project table is, in part, as follows:

```
INSERT INTO USER1.TPROJ
       (PROJNO,
        PRNAME,
        DEPTNO,
        DEPTMGR,
        PRSTAFF,
        PRSTDATE,
        PRENDATE,
        MAJPROJ)
  VALUES
       ('AD3100',
        'ADMINISTRATION SERVICES',
        'D01',
        '000010',
         6.5,
        '860101',
        '830201',
        '      ')

...and so on for the rest of the information for the table.
```

Inserting Information into the TEMPRACT Table

The INSERT statement for inserting information into the employee project account table is, in part, as follows:

```
INSERT INTO USER1.TEMPRACT
       (EMPNO,
        PROJNO,
        ACTNO,
        STARTDATE,
        ENDDATE,
        EMPTIME)
  VALUES
       ('000160',
        'MA2100',
         20,
        '860501',
        '860829',
         500)

...and so on for the rest of the information for the table.
```

Sample Tables

The following tables are the results of running the previous CREATE TABLE and INSERT statements:

Department Table (TDEPT)

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	COMPUTER SERVICE DIV.	000010	(blanks)
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER	(blanks)	A00
E01	SUPPORT SERVICES	000050	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E11	OPERATIONS	000090	E01
E21	SOFTWARE SUPPORT	000100	E01

Employee Table (TEMPL)

EMPNO	FIRSTNME	MIDINIT	LASTNAME	DEPTNO	PHONENO
000010	CHRISTINE	I	HAAS	A00	3978 ...
000020	MICHAEL	L	THOMPSON	B01	3476 ...
000030	SALLY	A	KWAN	C01	4738 ...
000050	JOHN	B	GEYER	E01	6789 ...
000060	IRVING	F	STERN	D11	6423 ...
000070	EVA	D	PULASKI	D21	7831 ...
000090	EILEEN	W	HENDERSON	E11	5498 ...
000100	THEODORE	Q	SPENSER	E21	0972 ...
000110	VICENZO	G	LUCCHESI	A00	3490 ...
000120	SEAN	(blanks)	O'CONNELL	A00	2167 ...
000130	DELORES	M	QUINTANA	C01	4578 ...
000140	HEATHER	A	NICHOLLS	C01	1793 ...
000150	BRUCE	(blanks)	ADAMSON	D11	4510 ...
000160	ELIZABETH	R	PIANKA	D11	3782 ...
000170	MASATOSHI	J	YOSHIMURA	D11	2890 ...
000180	MARILYN	S	SCOUTTEN	D11	1682 ...
000190	JAMES	H	WALKER	D11	2986 ...
000200	DAVID	(blanks)	BROWN	D11	4501 ...
000210	WILLIAM	T	JONES	D11	0942 ...
000220	JENNIFER	K	LUTZ	D11	0672 ...
000230	JAMES	J	JEFFERSON	D21	2094 ...
000240	SALVATORE	M	MARINO	D21	3780 ...
000250	DANIEL	S	SMITH	D21	0961 ...
000260	SYBIL	P	JOHNSON	D21	8953 ...
000270	MARIA	L	PEREZ	D21	9001 ...
000280	ETHEL	R	SCHNEIDER	E11	8997 ...

(Continued additional columns for each row are shown on the next page. The *EMPNO* column is repeated for reference only.)

Employee Table (TEMPL) (continued)

EMPNO	HIREDATE	JOBCODE	EDUCLVL	SEX	BRTHDATE	SALARY
000010 ...	750101	66	18	F	330814	52750
000020 ...	731010	61	18	M	480202	41250
000030 ...	750405	60	20	F	410511	38250
000050 ...	690817	58	16	M	450915	40175
000060 ...	730914	55	16	M	450707	32250
000070 ...	800930	56	16	F	530526	36170
000090 ...	700815	55	16	F	410515	29750
000100 ...	800619	54	14	M	561218	26150
000110 ...	680516	58	19	M	491105	46500
000120 ...	731205	58	14	M	421018	29250
000130 ...	710728	55	16	F	350915	23800
000140 ...	761215	55	18	F	460119	28420
000150 ...	720212	55	16	M	470517	25280
000160 ...	771011	54	17	F	550412	22250
000170 ...	780915	54	16	M	510105	24680
000180 ...	730707	53	17	F	490221	21340
000190 ...	740726	53	16	M	520625	20450
000200 ...	760303	55	16	M	410529	27740
000210 ...	790411	52	17	M	530223	18270
000220 ...	780829	55	18	F	480319	29840
000230 ...	761121	53	14	M	350530	22180
000240 ...	791205	55	17	M	540331	28760
000250 ...	791030	52	15	M	391112	19180
000260 ...	750911	52	16	F	361005	17250
000270 ...	800930	55	15	F	530526	27380
000280 ...	770324	54	17	F	360328	26250

Project Table (TPROJ)

PROJNO	PRNAME	DEPTNO	DEPTMGR	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
AD3100	ADMINISTRATION SERVICES	D01	000010	6.5	860101	830201	(blanks)
AD3110	GENERAL ADMIN SYSTEMS	D21	000070	6	871001	880215	AD3100
AD3111	PAYROLL PROGRAM-MING	D21	000230	2	880101	880401	AD3110
AD3112	PERSONNEL PRO-GRAMMING	D21	000250	1	870320	870601	AD3110
AD3113	ACCOUNT PROGRAM-MING	D21	000270	2	870901	880315	AD3110
MA2100	MFG AUTOMATION	D11	000060	12	860324	(blanks)	(blanks)
MA2110	MFG PROGRAMMING	E21	000100	3	870928	880219	MA2100
MA2112	ROBOT DESIGN	E01	000050	3	860106	881111	MA2110
MA2113	PROD CONTROL PROG	D11	000060	3	(blanks)	(blanks)	MA2100

Employee Project Account Table (TEMPRACT)

EMPNO	PROJNO	ACTNO	STARTDATE	ENDDATE	EMPTIME
000160	MA2100	20	860501	860829	500
000170	MA2100	20	860901	861231	500
000180	MA2100	20	870105	870430	650
000060	MA2100	10	870101	881101	500
000110	MA2100	20	880101	880301	400
000220	MA2112	50	871001	880615	900
000170	MA2112	70	870601	880102	100
000190	MA2112	70	880201	880601	100
000180	MA2113	70	870401	871215	400
000210	MA2113	80	870401	871215	500
000230	MA2113	70	870401	871215	300
000010	AD3100	10	880101	880701	500
000070	AD3110	10	880101	880201	100
000230	AD3111	60	880101	880315	100
000240	AD3111	70	880215	880915	500
000250	AD3112	60	880101	880201	100
000270	AD3113	60	880301	880401	100
000260	AD3113	70	880615	880701	80

Appendix C. SQLCODES

SQL does not communicate directly with the end user but rather returns error codes to the application program when an error occurs.

This appendix lists only SQLCODES. There are many SQL messages, but they are not listed here. Detailed descriptions of all SQL/400 messages, including SQLCODEs, are available online and can be displayed and printed from the Display Message Description display. You can access this display using the Display Message Description (DSPMSGD) CL command.

SQLCODEs are returned in the SQL communication area (SQLCA). The SQLCA is an area in the application program (defined by the application program) for the use of SQL. It is described in *SQL/400 Reference* manual.

Every SQLCODE has a corresponding message in message file QSQLMSG in library QSYS. The message ID for any SQLCODE is constructed by appending the absolute value (4 digits, padded with zeros) of the SQLCODE to SQL. For example, the message ID corresponding to SQLCODE −204 would be SQL0204. The replacement text for each SQL message is stored in SQLERRM in the SQLCA.

If SQL encounters an error while processing the statement, the SQLCODE is a *negative* number. If SQL encounters an exceptional but valid condition while processing your statement, the SQLCODE is a *positive* number. If your SQL statement is processed without encountering an error or exceptional condition, the return code is zero.

When running in debug mode, SQL places a message corresponding to the SQLCODE in the job log.

An application may also send the SQL message corresponding to any SQLCODE by specifying the message ID and the replacement text on the Retrieve Message (RTVMSG), Send Program Message (SNDPGMMSG), and Send User Message (SNDUSRMSG) CL commands. For more information on the CL commands, see the *CL Reference* manual.

SQLCODE Descriptions

In the following brief descriptions of the SQLCODEs, message data fields are identified by an ampersand (&) and a number (for example, &1). The replacement text for these fields is stored in SQLERRM in the SQLCA. More detailed cause and recovery information for any SQLCODE can be found by using the Display Message Description (DSPMSGD) CL command.

Positive SQLCODEs

N/A **SQLCODE 0**

Explanation: The SQL statement has run successfully. Check SQLWARN0 to ensure that it is blank. If it is blank, the statement was run successfully. If it is not blank, a warning condition exists. Check the other warning indicators to determine the particular warning condition. For example, if SQLWARN1 is not blank, a string has been truncated.

SQL0304 **SQLCODE +304**

Explanation: Conversion error in assignment to host variable &1.

SQL0802 **SQLCODE +802**

Explanation: Data conversion or data mapping error.

Negative SQLCODEs

SQL0010 **SQLCODE -10**

Explanation: String constant beginning &1 not delimited.

SQL0060 **SQLCODE -60**

Explanation: Value &1 for DECIMAL function not valid.

SQL0084 **SQLCODE -84**

Explanation: SQL statement not allowed.

SQL0100 **SQLCODE -100**

Explanation: Row not found.

SQL0101 **SQLCODE -101**

Explanation: SQL statement too long or complex.

SQL0103 **SQLCODE -103**

Explanation: Numeric literal &1 not valid.

SQL0104 **SQLCODE -104**

Explanation: Token &1 not valid. Valid tokens: &2.

SQL0105 **SQLCODE -105**

Explanation: Mixed string constant not valid.

SQL0106 SQLCODE -106

Explanation: Precision specified for FLOAT column not valid.

SQL0107 SQLCODE -107

Explanation: &1 too long. Maximum &2 characters.

SQL0109 SQLCODE -109

Explanation: &1 clause not allowed.

SQL0110 SQLCODE -110

Explanation: Hex literal beginning &1 not valid.

SQL0111 SQLCODE -111

Explanation: Column function does not include column name.

SQL0112 SQLCODE -112

Explanation: Operand of column function is another function.

SQL0113 SQLCODE -113

Explanation: Name &1 not allowed.

SQL0115 SQLCODE -115

Explanation: Comparison operator &1 not valid.

SQL0117 SQLCODE -117

Explanation: Statement inserts wrong number of values.

SQL0118 SQLCODE -118

Explanation: Table &1 in &2 also specified in FROM clause.

SQL0119 SQLCODE -119

Explanation: Column &1 in HAVING clause not in GROUP BY.

SQL0120 SQLCODE -120

Explanation: Column function use not valid.

SQL0121 SQLCODE -121

Explanation: Duplicate column name &1 in INSERT or UPDATE.

SQL0122 SQLCODE -122

Explanation: Column specified in SELECT list not valid.

SQL0125 SQLCODE -125

Explanation: ORDER BY column number &1 not valid.

SQL0129 SQLCODE -129

Explanation: Too many tables in SQL statement.

SQL0131 SQLCODE -131

Explanation: Comparison operator LIKE has operands not compatible.

SQL0132 SQLCODE -132

Explanation: LIKE predicate not valid.

SQL0134 SQLCODE -134

Explanation: Operand of MIN or MAX function exceeds 256 bytes.

SQL0136 SQLCODE -136

Explanation: ORDER BY or GROUP BY columns too long.

SQL0137 SQLCODE -137

Explanation: Result of concatenation too long.

SQL0138 SQLCODE -138

Explanation: Argument &1 of SUBSTR function out of range.

SQL0150 SQLCODE -150

Explanation: View or logical file &1 in &2 read only.

SQL0151 SQLCODE -151

Explanation: Column &3 in table &1 in &2 read only.

SQL0153 SQLCODE -153

Explanation: Column list required for CREATE VIEW.

SQL0154 SQLCODE -154

Explanation: UNION and UNION ALL for CREATE VIEW not valid.

SQL0155 SQLCODE -155

Explanation: View &1 in &2 not valid in FROM clause.

SQL0156 SQLCODE -156

Explanation: &1 in &2 not a table.

SQL0158 SQLCODE -158

Explanation: Number of columns specified not consistent.

SQL0159 SQLCODE -159

Explanation: &1 in &2 not correct type.

SQL0170 SQLCODE -170

Explanation: Number of arguments for function &1 not valid.

SQL0171 SQLCODE -171

Explanation: Argument of function &2 is not valid.

SQL0198 SQLCODE -198

Explanation: SQL statement empty or blank.

SQL0199 SQLCODE -199

Explanation: Keyword &1 not expected. Valid tokens: &2.

SQL0203 SQLCODE -203

Explanation: Column &1 in more than one table.

SQL0204 SQLCODE -204

Explanation: Object &1 in &2 type *&3 not found.

SQL0205 SQLCODE -205

Explanation: Column &1 not in table &2.

SQL0206 SQLCODE -206

Explanation: Column &1 not in specified tables.

SQL0207 SQLCODE -207

Explanation: ORDER BY clause with UNION operator not valid.

SQL0208 SQLCODE -208

Explanation: ORDER BY column &1 not in results table.

SQL0301 SQLCODE -301

Explanation: Input host variable &1 not valid.

SQL0302 SQLCODE -302

Explanation: Conversion error on input host variable &1.

SQL0303 SQLCODE -303

Explanation: Host variable &1 not compatible with SELECT item.

SQL0304 SQLCODE -304

Explanation: Conversion error in assignment to host variable &1.

SQL0305 SQLCODE -305

Explanation: Indicator variable required.

SQL0309 SQLCODE -309

Explanation: Indicator variable contains null value.

SQL0311 SQLCODE -311

Explanation: Variable length host variable contains length less than zero.

SQL0312 SQLCODE -312

Explanation: Host variable &1 not defined or not usable.

SQL0313 SQLCODE -313

Explanation: Number host variables not valid.

SQL0401 SQLCODE -401

Explanation: Comparison operator &1 operands not compatible.

SQL0402 SQLCODE -402

Explanation: &1 use not valid.

SQL0404 SQLCODE -404

Explanation: Value for column &1 too long.

SQL0405 SQLCODE -405

Explanation: Literal &1 out of range.

SQL0406 SQLCODE -406

Explanation: Conversion error on assignment to column &1.

SQL0407 SQLCODE -407

Explanation: INSERT or UPDATE value is null.

SQL0408 SQLCODE -408

Explanation: INSERT or UPDATE value for column &1 not compatible.

SQL0410 SQLCODE -410

Explanation: Floating point literal &1 not valid.

SQL0414 SQLCODE -414

Explanation: Numeric column &1 not valid in LIKE predicate.

SQL0415 SQLCODE -415

Explanation: UNION operands not compatible.

SQL0417 SQLCODE -417

Explanation: Combination of parameter markers not valid.

SQL0418 SQLCODE -418

Explanation: Usage of parameter marker is not valid.

SQL0419 SQLCODE -419

Explanation: Negative scale not valid.

SQL0421 SQLCODE -421

Explanation: Number of UNION operands not equal.

SQL0501 SQLCODE -501

Explanation: Cursor &1 not open.

SQL0502 SQLCODE -502

Explanation: Cursor &1 already open.

SQL0503 SQLCODE -503

Explanation: Column &3 can not be updated.

SQL0504 SQLCODE -504

Explanation: Cursor &1 not declared.

SQL0507 SQLCODE -507

Explanation: Cursor &1 not open.

SQL0508 SQLCODE -508

Explanation: Cursor &1 not positioned on locked row.

SQL0509 SQLCODE -509

Explanation: Table &2 in &3 not same as table in cursor &1.

SQL0510 SQLCODE -510

Explanation: View or logical file &1 in &2 read only.

SQL0511 SQLCODE -511

Explanation: FOR UPDATE OF clause not valid.

SQL0514 SQLCODE -514

Explanation: Prepared statement &2 not found.

SQL0516 SQLCODE -516

Explanation: Prepared statement &1 not found.

SQL0517 SQLCODE -517

Explanation: Prepared statement &2 not SELECT statement.

SQL0518 SQLCODE -518

Explanation: Prepared statement &1 not found.

SQL0519 SQLCODE -519

Explanation: Prepared statement &1 in use.

SQL0551 SQLCODE -551

Explanation: Not authorized to object &1 in &2 type *&3.

SQL0552 SQLCODE -552

Explanation: Not authorized to &1.

SQL0556 SQLCODE -556

Explanation: Revoke of privilege not valid.

SQL0601 SQLCODE -601

Explanation: Object &1 in &2 type *&3 already exists.

SQL0602 SQLCODE -602

Explanation: More the 120 columns specified for CREATE INDEX.

SQL0603 SQLCODE -603

Explanation: Unique index can not be created because of duplicate keys.

SQL0604 SQLCODE -604

Explanation: Attributes of column not valid.

SQL0607 SQLCODE -607

Explanation: Operation not allowed on system table &1 in &2.

SQL0612 SQLCODE -612

Explanation: &1 is a duplicate column name.

SQL0614 SQLCODE -614

Explanation: Length of columns for CREATE INDEX too long.

SQL0637 SQLCODE -637

Explanation: Duplicate &1 keyword.

SQL0802 SQLCODE -802

Explanation: Data conversion or data mapping error.

SQL0803 SQLCODE -803

Explanation: Duplicate key value specified.

SQL0804 SQLCODE -804

Explanation: SQLDA not valid.

SQL0811 SQLCODE -811

Explanation: Embedded SELECT resulted in more than one row.

SQL0822 SQLCODE -822

Explanation: SQLDA contains address not valid.

SQL0840 SQLCODE -840

Explanation: Number of selected items exceeds 8000.

SQL0901 SQLCODE -901

Explanation: Function check.

SQL0904 SQLCODE -904

Explanation: Resource limit exceeded.

SQL0906 SQLCODE -906

Explanation: Operation not performed because previous error.

SQL0913 SQLCODE -913

Explanation: Row or object &1 in &2 type *&3 in use.

SQL5001 SQLCODE -5001

Explanation: Column qualifier &1 is undefined.

SQL5002 SQLCODE -5002

Explanation: Library must be specified for table &1.

SQL5003 SQLCODE -5003

Explanation: DDL statement used with commitment control.

SQL5004 SQLCODE -5004

Explanation: CREATE VIEW not valid with USER or LENGTH specified.

SQL5005 SQLCODE -5005

Explanation: Operator &1 not consistent with operands.

SQL5006 SQLCODE -5006

Explanation: Duplicate table designator &1 is not valid.

SQL5016 SQLCODE -5016

Explanation: Table name &1 not valid for naming option.

SQL5017 SQLCODE -5017

Explanation: Too many users specified for GRANT or REVOKE.

SQL5019 SQLCODE -5019

Explanation: Empty string operand not valid.

SQL5021 SQLCODE -5021

Explanation: FOR UPDATE OF column &1 also in ORDER BY.

SQL5022 SQLCODE -5022

Explanation: FOR DATA clause not valid for specified type.

SQL7001 SQLCODE -7001

Explanation: File &1 in &2 not data base file.

SQL7002 SQLCODE -7002

Explanation: Override parameter not valid.

SQL7003 SQLCODE -7003

Explanation: File &1 in &2 has more than one format.

SQL7004 SQLCODE -7004

Explanation: &1 not a data base.

SQL7005 SQLCODE -7005

Explanation: FOR MIXED DATA not allowed.

SQL7006 SQLCODE -7006

Explanation: Cannot drop data base &1.

SQL7007 SQLCODE -7007

Explanation: COMMIT or ROLLBACK not valid.

SQL7008 SQLCODE -7008

Explanation: Table &1 in &2 not valid for operation.

SQL7009 SQLCODE -7009

Explanation: USER specified, but user profile name &1 too long.

SQL7010 SQLCODE -7010

Explanation: Logical File &1 not valid for CREATE VIEW.

SQL7011 SQLCODE -7011

Explanation: &1 in &2 not table, view, or physical file.

SQL7012 SQLCODE -7012

Explanation: Privilege not valid for table or view &1 in &2.

Appendix D. Sample Programs Using SQL/400 Statements

This appendix contains a sample application showing how to code SQL statements in each of the three languages supported by SQL/400: COBOL, PL/I, and RPG.

The sample application adjusts the estimated employee hours for project numbers starting with MA. The application uses the sample tables in Appendix B.

Each sample program produces the same report, which is shown at the end of this appendix. The first part of the report shows the updated estimated employee hours. The second part shows the previous total and the current total of estimated hours by project.

The following notes apply to all three sample programs:

SQL statements can be entered in upper or lower case.

A This host language statement retrieves the external definitions for the SQL table TEMPRACT. These definitions can be used as host variables or as a host structure.

Note: In RPG, field names in an externally described structure that are longer than 6 characters must be renamed.

B The SQL Communications Area (SQLCA) is required for all SQL programs. The SQL INCLUDE SQLCA statement can be used to include the SQLCA for PL/I and COBOL programs. For RPG programs, the SQL precompiler automatically places the SQLCA data structure into the source at the end of the I specification section.

C This SQL WHENEVER statement defines the host language label to which control is passed if an SQLERROR (SQLCODE < 0) occurs in an SQL statement. This WHENEVER SQLERROR statement applies to all the following SQL statements until the next WHENEVER SQLERROR statement is encountered.

D This SQL UPDATE statement updates the *EMPTIME* column, which contains the estimated hours by the rate in the host variable PERCENTAGE (PERCNT for RPG). The updated rows are those that have MA as the first two characters of the *PROJNO* column.

E This SQL COMMIT statement commits the changes made by the SQL UPDATE statement. Record locks on all changed rows are released.

Note: The program was precompiled using COMMIT(*CHG).

F This SQL DECLARE CURSOR statement defines cursor C1, which returns all columns of each row from the TEMPRACT table in which MA are the first two characters of the *PROJNO* column. Rows are returned in ascending order by employee number (*EMPNO* column).

G This SQL OPEN statement opens cursor C1 so that the rows can be fetched.

H This SQL WHENEVER statement defines the host language label to which control is passed when all rows are fetched (SQLCODE = 100).

I This SQL FETCH statement returns all columns for cursor C1 and places the returned values into the corresponding elements of the host structure. See Note A.

J After all rows are fetched, control is passed to this label. The SQL CLOSE statement closes cursor C1.

K This SQL DECLARE CURSOR statement defines cursor C2, which joins the two tables TEMPRACT and TPROJ. The results are grouped by the columns *PROJNO* and *PRNAME*. The COUNT function returns the number of rows in each group. The SUM functions calculate the total of previously estimated hours and the total of current estimated hours. The ORDER BY 1 clause specifies that rows are retrieved based on the contents of the first results column (*TEMPRACT.PROJNO*).

L This SQL FETCH statement returns the results columns for cursor C2 and places the returned values into the corresponding elements of the host structure described by the program.

M This SQL WHENEVER statement with the CONTINUE option causes processing to continue to the next statement regardless if an error occurs on the SQL ROLLBACK statement. Errors are not expected on the SQL ROLLBACK statement; however, this prevents the program from going into a loop if an error does occur.

N This SQL ROLLBACK statement restores the table to its original condition if an error occurred during the update.

```
5728ST1 R01 M02 880901                    IBM SQL/400       CBLEX                    05/05/88 09:39:10   Page   1
Source type.............COBOL
Program name............CLR/CBLEX
Source file.............*LIBL/QLBLSRC
Member..................*PGM
Options.................*SRC      *XREF
INCLUDE file............*LIBL/*SRCFILE
Commit..................*CHG
Generation level........0
Printer file............*LIBL/SQLPRT
Text....................*SRCMBRTXT
Source member changed on 05/05/88  09:38:55
                                                                                                    RSLS773-0
```

Figure D-1 (Part 1 of 9). Sample COBOL Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       CBLEX                    05/05/88 09:39:10   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
   1         IDENTIFICATION DIVISION.                                                    100
   2                                                                                     200
   3         PROGRAM-ID.  CBLEX.                                                         300
   4         ENVIRONMENT DIVISION.                                                       400
   5         CONFIGURATION SECTION.                                                      500
   6         SOURCE-COMPUTER. IBM-AS400.                                                 600
   7         OBJECT-COMPUTER. IBM-AS400.                                                 700
   8         INPUT-OUTPUT SECTION.                                                       800
   9                                                                                     900
  10         FILE-CONTROL.                                                              1000
  11             SELECT PRINTFILE ASSIGN TO PRINTER-QPRINT                              1100
  12                ORGANIZATION IS SEQUENTIAL.                                         1200
  13                                                                                    1300
  14         DATA DIVISION.                                                             1400
  15                                                                                    1500
  16         FILE SECTION.                                                              1600
  17                                                                                    1700
  18         FD  PRINTFILE                                                              1800
  19             BLOCK CONTAINS 1 RECORDS                                               1900
  20             LABEL RECORDS ARE OMITTED.                                             2000
  21         01  PRINT-RECORD PIC X(132).                                              2100
  22                                                                                    2200
  23         WORKING-STORAGE SECTION.                                                   2300
  24         77  PROJID PIC X(3).                                                      2400
  25         77  PERCENTAGE PIC S999V99 COMP-3.                                        2500
  26                                                                                    2600
  27        ****************************************************************          2700
  28        *  Structure for report 1.                                     *          2800
  29        ****************************************************************          2900
  30                                                                                    3000
  31         01  RPT1.                                                                 3100
  32    A        COPY DDS-TEMPRACT OF USER1-TEMPRACT.                                  3200
  33                                                                                    3300
  34        ****************************************************************          3400
  35        *  Structure for report 2.                                     *          3500
  36        ****************************************************************          3600
  37                                                                                    3700
  38         01  RPT2.                                                                 3800
  39             15  PROJNO PIC X(6).                                                  3900
  40             15  PROJECT-NAME PIC X(36).                                           4000
  41             15  EMPLOYEE-COUNT PIC S9(4) COMP-4.                                  4100
  42             15  OLD-TOTAL-TIME PIC S9(6)V99 COMP-3.                               4200
  43             15  NEW-TOTAL-TIME PIC S9(6)V99 COMP-3.                               4300
  44                                                                                    4400
  45    B        EXEC SQL                                                            4500
  46                  INCLUDE SQLCA                                                   4600
  47             END-EXEC.                                                          4700
  48         77  CODE-EDIT PIC ---99.                                              4800
  49                                                                                    4900
  50        ****************************************************************          5000
  51        *  Headers for reports.                                        *          5100
  52        ****************************************************************          5200
  53                                                                                    5300
  54         01  RPT1-HEADERS.                                                         5400
  55             05  RPT1-HEADER1 PIC X(132)                                           5500
  56                       VALUE "UPDATED EMPLOYEE PROJECT ACCOUNT DATA".               5600
  57             05  RPT1-HEADER2.                                                     5700
  58                 10  FILLER PIC X(10) VALUE "EMPLOYEE".                            5800
  59                 10  FILLER PIC X(9) VALUE "PROJECT".                              5900
  60                 10  FILLER PIC X(9) VALUE "ACCOUNT".                              6000
                                                                                                    RSLS774-0
```

Figure D-1 (Part 2 of 9). Sample COBOL Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       CBLEX                    05/05/88 09:39:10   Page     3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8    SEQNBR  Last change
   61                10  FILLER PIC X(104) VALUE "EMPLOYEE".                                    6100
   62             05  RPT1-HEADER3.                                                            6200
   63                 10  FILLER PIC X(10) VALUE " NUMBER".                                    6300
   64                 10  FILLER PIC X(9) VALUE "NUMBER".                                      6400
   65                 10  FILLER PIC X(9) VALUE "NUMBER".                                      6500
   66                 10  FILLER PIC X(104) VALUE " HOURS".                                    6600
   67         01  RPT2-HEADERS.                                                               6700
   68             05  RPT2-HEADER1.                                                           6800
   69                 10  FILLER PIC X(21) VALUE SPACES.                                       6900
   70                 10  FILLER PIC X(111)                                                    7000
   71                       VALUE "ACCUMULATED STATISTICS BY PROJECT".                        7100
   72             05  RPT2-HEADER2.                                                           7200
   73                 10  FILLER PIC X(9) VALUE "PROJECT".                                     7300
   74                 10  FILLER PIC X(38) VALUE SPACES.                                       7400
   75                 10  FILLER PIC X(11) VALUE "NUMBER OF".                                  7500
   76                 10  FILLER PIC X(10) VALUE "PREVIOUS".                                   7600
   77                 10  FILLER PIC X(65) VALUE " CURRENT".                                   7700
   78             05  RPT2-HEADER3.                                                           7800
   79                 10  FILLER PIC X(9) VALUE "NUMBER".                                      7900
   80                 10  FILLER PIC X(38) VALUE "PROJECT NAME".                               8000
   81                 10  FILLER PIC X(11) VALUE "EMPLOYEES".                                  8100
   82                 10  FILLER PIC X(10) VALUE " HOURS".                                     8200
   83                 10  FILLER PIC X(65) VALUE "  HOURS".                                    8300
   84         01  RPT1-DATA.                                                                  8400
   85             05  FILLER PIC X VALUE SPACE.                                               8500
   86             05  EMPNO PIC X(6).                                                         8600
   87             05  FILLER PIC XXX VALUE SPACES.                                            8700
   88             05  PROJNO PIC X(6).                                                        8800
   89             05  FILLER PIC X(4) VALUE SPACES.                                           8900
   90             05  ACTNO PIC ZZZ99.                                                        9000
   91             05  FILLER PIC X(3) VALUE SPACES.                                           9100
   92             05  EMPTIME PIC ZZZZ9.99.                                                   9200
   93             05  FILLER PIC X(96) VALUE SPACES.                                          9300
   94         01  RPT2-DATA.                                                                  9400
   95             05  PROJNO PIC X(6).                                                        9500
   96             05  FILLER PIC XXX VALUE SPACES.                                            9600
   97             05  PROJECT-NAME PIC X(36).                                                 9700
   98             05  FILLER PIC X(4) VALUE SPACES.                                           9800
   99             05  EMPLOYEE-COUNT PIC ZZZ9.                                                9900
  100             05  FILLER PIC X(5) VALUE SPACES.                                          10000
  101             05  OLD-TOTAL-TIME PIC ZZZZ9.99.                                           10100
  102             05  FILLER PIC XX VALUE SPACES.                                            10200
  103             05  NEW-TOTAL-TIME PIC ZZZZ9.99.                                           10300
  104             05  FILLER PIC X(56) VALUE SPACES.                                         10400
  105                                                                                         10500
  106         PROCEDURE DIVISION.                                                             10600
  107                                                                                         10700
  108         A000-MAIN.                                                                      10800
  109             MOVE 0.06 TO PERCENTAGE.                                                    10900
  110             MOVE "MA%" TO PROJID.                                                       11000
  111             OPEN OUTPUT PRINTFILE.                                                      11100
  112                                                                                         11200
  113        ****************************************************************               11300
  114        * Update the selected projects by the new percentage. If an   *               11400
  115        * error occurs during the updat, ROLLBACK the changes,        *               11500
  116        ****************************************************************               11600
  117                                                                                         11700
  118         C      EXEC SQL                                                                11800
  119                    WHENEVER SQLERROR GO TO E010-UPDATE-ERROR                          11900
  120             END-EXEC.                                                                  12000
                                                                                                RSLS775-0
```

Figure D-1 (Part 3 of 9). Sample COBOL Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       CBLEX                    05/05/88 09:39:10   Page     4
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8    SEQNBR  Last change
  121             EXEC SQL                                                            12100
  122   D              UPDATE USER1/TEMPRACT                                        12200
  123                    SET EMPTIME = EMPTIME * (1+:PERCENTAGE)                    12300
  124                    WHERE PROJNO LIKE :PROJID                                  12400
  125             END-EXEC.                                                         12500
  126                                                                               12600
  127         ***************************************************************       12700
  128         *  Commit changes.                                            *       12800
  129         ***************************************************************       12900
  130                                                                               13000
  131             EXEC SQL                                                          13100
  132   E              COMMIT                                                       13200
  133             END-EXEC.                                                         13300
  134                                                                               13400
  135             EXEC SQL                                                          13500
  136                  WHENEVER SQLERROR GO TO E020-REPORT-ERROR                    13600
  137             END-EXEC.                                                         13700
  138                                                                               13800
  139         ***************************************************************       13900
  140         *  Report the updated statistics for each employee assigned to*       14000
  141         *  the selected projects.                                     *       14100
  142         ***************************************************************       14200
  143                                                                               14300
  144         ***************************************************************       14400
  145         *  Write out the header for Report 1.                         *       14500
  146         ***************************************************************       14600
  147                                                                               14700
  148             write print-record from rpt1-header1                              14800
  149                   before advancing 2 lines.                                   14900
  150             write print-record from rpt1-header2                              15000
  151                   before advancing 1 line.                                    15100
  152             write print-record from rpt1-header3                              15200
  153                   before advancing 2 lines.                                   15300
  154             exec sql                                                          15400
  155   F              declare c1 cursor for                                        15500
  156                     select *                                                  15600
  157                     from user1/tempract                                       15700
  158                     where tempract.projno like :projid                        15800
  159                     order by empno                                            15900
  160             end-exec.                                                         16000
  161             EXEC SQL                                                          16100
  162   G              OPEN C1                                                      16200
  163             END-EXEC.                                                         16300
  164                                                                               16400
  165             PERFORM B000-GENERATE-REPORT1 THRU B010-GENERATE-REPORT1-EXIT      16500
  166                 UNTIL SQLCODE NOT EQUAL TO ZERO.                              16600
  167                                                                               16700
  168          A100-DONE1.                                                          16800
  169   J         EXEC SQL                                                         16900
  170                  CLOSE C1                                                     17000
  171             END-EXEC.                                                         17100
  172                                                                               17200
  173         ***************************************************************       17300
  174         *  For each project selected, generate a report containing the*       17400
  175         *  project number, project name, the old total of employee    *       17500
  176         *  hours, and the new total of employee hours for each        *       17600
  177         *  project.                                                   *       17700
  178         ***************************************************************       17800
  179                                                                               17900
  180                                                                               18000
```

Note: H and I are located on Part 5 of this figure.

RSLS776-0

Figure D-1 (Part 4 of 9). Sample COBOL Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       CBLEX                    05/05/88 09:39:10   Page     5
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8    SEQNBR  Last change
  181       ************************************************************              18100
  182       *  Write out the header for Report 2.                      *              18200
  183       ************************************************************              18300
  184                                                                                 18400
  185           MOVE SPACES TO PRINT-RECORD.                                          18500
  186           WRITE PRINT-RECORD BEFORE ADVANCING 2 LINES.                          18600
  187           WRITE PRINT-RECORD FROM RPT2-HEADER1                                  18700
  188                 BEFORE ADVANCING 2 LINES.                                       18800
  189           WRITE PRINT-RECORD FROM RPT2-HEADER2                                  18900
  190                 BEFORE ADVANCING 1 LINE.                                        19000
  191           WRITE PRINT-RECORD FROM RPT2-HEADER3                                  19100
  192                 BEFORE ADVANCING 2 LINES.                                       19200
  193                                                                                 19300
  194           EXEC SQL                                                              19400
  195      K         DECLARE C2 CURSOR FOR                                          19500
  196                   SELECT TEMPRACT.PROJNO, PRNAME, COUNT(*),                   19600
  197                          SUM(EMPTIME/(1.0+:PERCENTAGE)),SUM(EMPTIME)          19700
  198                   FROM USER1/TEMPRACT, USER1/TPROJ                            19800
  199                   WHERE TEMPRACT.PROJNO=TPROJ.PROJNO                          19900
  200                   GROUP BY TEMPRACT.PROJNO, PRNAME                            20000
  201                   HAVING TEMPRACT.PROJNO LIKE :PROJID                         20100
  202                   ORDER BY 1                                                  20200
  203           END-EXEC.                                                           20300
  204           EXEC SQL                                                            20400
  205                OPEN C2                                                        20500
  206           END-EXEC.                                                           20600
  207                                                                               20700
  208           PERFORM C000-GENERATE-REPORT2 THRU C010-GENERATE-REPORT2-EXIT       20800
  209                UNTIL SQLCODE NOT EQUAL TO ZERO.                               20900
  210                                                                               21000
  211       A200-DONE2.                                                           21100
  212           EXEC SQL                                                          21200
  213                CLOSE C2                                                     21300
  214           END-EXEC.                                                         21400
  215                                                                             21500
  216       ************************************************************          21600
  217       *  All done.                                               *          21700
  218       ************************************************************          21800
  219                                                                             21900
  220       A900-MAIN-EXIT.                                                       22000
  221           CLOSE PRINTFILE.                                                  22100
  222           STOP RUN.                                                         22200
  223                                                                             22300
  224       ************************************************************          22400
  225       *  Fetch and write the rows to PRINTFILE.                  *          22500
  226       ************************************************************          22600
  227                                                                             22700
  228       B000-GENERATE-REPORT1.                                                22800
  229           EXEC SQL                                                          22900
  230      H         WHENEVER NOT FOUND GO TO A100-DONE1                          23000
  231           END-EXEC.                                                         23100
  232           EXEC SQL                                                          23200
  233      I         FETCH C1 INTO :TEMPRACT                                      23300
  234           END-EXEC.                                                         23400
  235           MOVE CORRESPONDING TEMPRACT TO RPT1-DATA.                         23500
  236           WRITE PRINT-RECORD FROM RPT1-DATA                                 23600
  237                 BEFORE ADVANCING 1 LINE.                                    23700
  238                                                                             23800
  239       B010-GENERATE-REPORT1-EXIT.                                           23900
  240           EXIT.                                                             24000
                                                                                  RSLS777-0
```

Figure D-1 (Part 5 of 9). Sample COBOL Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       CBLEX                    05/05/88 09:39:10   Page     6
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8    SEQNBR  Last change
  241                                                                                          24100
  242         ************************************************************                     24200
  243         *  Fetch and write the rows to PRINTFILE.                  *                     24300
  244         ************************************************************                     24400
  245                                                                                          24500
  246          C000-GENERATE-REPORT2.                                                          24600
  247              EXEC SQL                                                                    24700
  248                   WHENEVER NOT FOUND GO TO A200-DONE2                                    24800
  249              END-EXEC.                                                                   24900
  250              EXEC SQL                                                                    25000
  251   L               FETCH C2 INTO :RPT2                                                    25100
  252              END-EXEC.                                                                   25200
  253              MOVE CORRESPONDING RPT2 TO RPT2-DATA.                                       25300
  254              WRITE PRINT-RECORD FROM RPT2-DATA                                           25400
  255                    BEFORE ADVANCING 1 LINE.                                              25500
  256                                                                                          25600
  257          C010-GENERATE-REPORT2-EXIT.                                                     25700
  258              EXIT.                                                                       25800
  259                                                                                          25900
  260         ************************************************************                     26000
  261         *  Error occured while updating table.  Inform user and    *                     26100
  262         *  rollback changes.                                       *                     26200
  263         ************************************************************                     26300
  264                                                                                          26400
  265          E010-UPDATE-ERROR.                                                              26500
  266              EXEC SQL                                                                    26600
  267   M               WHENEVER SQLERROR CONTINUE                                             26700
  268              END-EXEC.                                                                   26800
  269              MOVE SQLCODE TO CODE-EDIT.                                                  26900
  270              STRING "*** ERROR Occurred while updating table.  SQLCODE="                 27000
  271                     CODE-EDIT DELIMITED BY SIZE INTO PRINT-RECORD.                       27100
  272              WRITE PRINT-RECORD.                                                         27200
  273              EXEC SQL                                                                    27300
  274   N               ROLLBACK                                                               27400
  275              END-EXEC.                                                                   27500
  276              STOP RUN.                                                                   27600
  277                                                                                          27700
  278         ************************************************************                     27800
  279         *  Error occured while generating reports.  Inform user and *                     27900
  280         *  exit.                                                   *                     28000
  281         ************************************************************                     28100
  282                                                                                          28200
  283          E020-REPORT-ERROR.                                                              28300
  284              MOVE SQLCODE TO CODE-EDIT.                                                  28400
  285              STRING "*** ERROR Occurred while generating reports.  SQLCODE               28500
  286         -           "=" CODE-EDIT DELIMITED BY SIZE INTO PRINT-RECORD.                   28600
  287              WRITE PRINT-RECORD.                                                         28700
  288              STOP RUN.                                                                   28800
                          * * * * *  E N D  O F  S O U R C E  * * * * *
```

RSLS778-0

Figure D-1 (Part 6 of 9). Sample COBOL Program Using SQL Statements

```
5728ST1 R01 M02 880901                IBM SQL/400       CBLEX                    05/05/88 09:39:10   Page     7
                                      CROSS REFERENCE
Data Names                 Define    Reference
'ACTNO'                       122      SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN 'USER1'.'TEMPRACT'
'C1'                          155      CURSOR
                                       162 170 233
'C2'                          195      CURSOR
                                       205 213 251
'DEPTMGR'                     198      CHARACTER(6) COLUMN (NOT NULL) IN 'USER1'.'TPROJ'
'DEPTNO'                      198      CHARACTER(3) COLUMN (NOT NULL) IN 'USER1'.'TPROJ'
'EMPNO'                       122      CHARACTER(6) COLUMN (NOT NULL) IN 'USER1'.'TEMPRACT'
'EMPNO'                       ****     COLUMN
                                       159
'EMPTIME'                     ****     COLUMN
                                       123 123 197 197
'EMPTIME'                     122      DECIMAL(5,2) COLUMN (NOT NULL) IN 'USER1'.'TEMPRACT'
'ENDDATE'                     122      CHARACTER(6) COLUMN (NOT NULL) IN 'USER1'.'TEMPRACT'
'MAJPROJ'                     198      CHARACTER(6) COLUMN (NOT NULL) IN 'USER1'.'TPROJ'
'PRENDATE'                    198      CHARACTER(6) COLUMN (NOT NULL) IN 'USER1'.'TPROJ'
'PRNAME'                      ****     COLUMN
                                       196 200
'PRNAME'                      198      CHARACTER(36) COLUMN (NOT NULL) IN 'USER1'.'TPROJ'
'PROJNO'                      ****     COLUMN
                                       124
'PROJNO'                      122      CHARACTER(6) COLUMN (NOT NULL) IN 'USER1'.'TEMPRACT'
'PROJNO'                      ****     COLUMN IN 'USER1'.'TEMPRACT'
                                       158
'PROJNO'                      ****     COLUMN IN 'USER1'.'TEMPRACT'
                                       196
'PROJNO'                      ****     COLUMN IN 'USER1'.'TEMPRACT'
                                       199
'PROJNO'                      ****     COLUMN IN 'USER1'.'TPROJ'
                                       199
'PROJNO'                      ****     COLUMN IN 'USER1'.'TEMPRACT'
                                       200
'PROJNO'                      ****     COLUMN IN 'USER1'.'TEMPRACT'
                                       201
'PROJNO'                      198      CHARACTER(6) COLUMN (NOT NULL) IN 'USER1'.'TPROJ'
'PRSTAFF'                     198      DECIMAL(5,2) COLUMN (NOT NULL) IN 'USER1'.'TPROJ'
'PRSTDATE'                    198      CHARACTER(6) COLUMN (NOT NULL) IN 'USER1'.'TPROJ'
'STARTDATE'                   122      CHARACTER(6) COLUMN (NOT NULL) IN 'USER1'.'TEMPRACT'
'TEMPRACT'                    ****     TABLE IN 'USER1'
                                       122 157 158 196 198 199 200 201
'TPROJ'                       ****     TABLE IN 'USER1'
                                       198 199
'USER1'                       ****     DATABASE
                                       122 157 198 198

                                                                                                  RSLS779-0
```

Figure D-1 (Part 7 of 9). Sample COBOL Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       CBLEX          05/05/88 09:39:10   Page     8
                                          CROSS REFERENCE
ACTNO                          32       SMALL INTEGER PRECISION(4,0) IN TEMPRACT
ACTNO                          90       IN RPT1-DATA
A100-DONE1                     ****     LABEL
                                        230
A200-DONE2                     ****     LABEL
                                        248
CODE-EDIT                      48
EMPLOYEE-COUNT                 41       SMALL INTEGER PRECISION(4,0) IN RPT2
EMPLOYEE-COUNT                 99       IN RPT2-DATA
EMPNO                          32       CHARACTER(6) IN TEMPRACT
EMPNO                          86       CHARACTER(6) IN RPT1-DATA
EMPTIME                        32       DECIMAL(5,2) IN TEMPRACT
EMPTIME                        92       IN RPT1-DATA
ENDDATE                        32       CHARACTER(6) IN TEMPRACT
E010-UPDATE-ERROR              ****     LABEL
                                        119
E020-REPORT-ERROR              ****     LABEL
                                        136
NEW-TOTAL-TIME                 43       DECIMAL(8,2) IN RPT2
NEW-TOTAL-TIME                 103      IN RPT2-DATA
OLD-TOTAL-TIME                 42       DECIMAL(8,2) IN RPT2
OLD-TOTAL-TIME                 101      IN RPT2-DATA
PERCENTAGE                     25       DECIMAL(5,2)
                                        123 197
PRINT-RECORD                   21       CHARACTER(132)
PROJECT-NAME                   40       CHARACTER(36) IN RPT2
PROJECT-NAME                   97       CHARACTER(36) IN RPT2-DATA
PROJID                         24       CHARACTER(3)
                                        124 158 201
PROJNO                         32       CHARACTER(6) IN TEMPRACT
PROJNO                         39       CHARACTER(6) IN RPT2
PROJNO                         88       CHARACTER(6) IN RPT1-DATA
PROJNO                         95       CHARACTER(6) IN RPT2-DATA
RPT1                           31
RPT1-DATA                      84
RPT1-HEADERS                   54
RPT1-HEADER1                   55       CHARACTER(132) IN RPT1-HEADERS
RPT1-HEADER2                   57       IN RPT1-HEADERS
                                                                                        RSLS780-0
```

Figure D-1 (Part 8 of 9). Sample COBOL Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       CBLEX          05/05/88 09:39:10   Page     9
                                          CROSS REFERENCE
RPT1-HEADER3                   62       IN RPT1-HEADERS
RPT2                           38       STRUCTURE
                                        251
RPT2-DATA                      94
RPT2-HEADERS                   67
RPT2-HEADER1                   68       IN RPT2-HEADERS
RPT2-HEADER2                   72       IN RPT2-HEADERS
RPT2-HEADER3                   78       IN RPT2-HEADERS
STARTDATE                      32       CHARACTER(6) IN TEMPRACT
TEMPRACT                       32       STRUCTURE IN RPT1
No errors found in source
  288 Source records processed
                              * * * * *  E N D  O F  L I S T I N G  * * * * *
                                                                                        RSLS781-0
```

Figure D-1 (Part 9 of 9). Sample COBOL Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       PLIEX                    05/05/88 09:37:30   Page   1
Source type.............PLI
Program name............CLR/PLIEX
Source file.............*LIBL/QPLISRC
Member..................*PGM
Options.................*SRC      *XREF
INCLUDE file............*LIBL/*SRCFILE
Commit..................*CHG
Generation level........0
Margins.................*SRCFILE
Printer file............*LIBL/SQLPRT
Text....................*SRCMBRTXT
Source member changed on 05/05/88  09:35:40

                                                                                                      RSLS782-0
```

Figure D-2 (Part 1 of 6). Sample PL/I Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400      PLIEX                    05/05/88 09:37:30   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
    1   PLIEX: PROC;                                                                    100
    2                                                                                   200
    3     DCL PROJID CHAR(3);                                                           300
    4     DCL PERCENTAGE FIXED DECIMAL(5,2);                                            400
    5                                                                                   500
    6     /* File declaration for sysprint */                                           600
    7     DCL SYSPRINT FILE EXTERNAL OUTPUT STREAM PRINT;                               700
    8                                                                                   800
    9     /* Structure for report 1 */                                                  900
   10     DCL 1 RPT1,                                                                  1000
   11 A         %INCLUDE TEMPRACT(TEMPRACT,RECORD);                                    1100
   12                                                                                  1200
   13     /* Structure for report 2 */                                                 1300
   14     DCL 1 RPT2,                                                                  1400
   15           15 PROJNO CHAR(6),                                                     1500
   16           15 PROJECTXNAME CHAR(36),                                              1600
   17           15 EMPLOYEEXCOUNT FIXED BIN(15),                                       1700
   18           15 OLDXTOTALXTIME FIXED DECIMAL(8,2),                                  1800
   19           15 NEWXTOTALXTIME FIXED DECIMAL(8,2);                                  1900
   20                                                                                  2000
   21 B  EXEC SQL INCLUDE SQLCA;                                                       2100
   22                                                                                  2200
   23     PERCENTAGE = 0.06;                                                           2300
   24     PROJID = 'MA%';                                                              2400
   25     OPEN FILE(SYSPRINT);                                                         2500
   26                                                                                  2600
   27     /* Update the selected projects by the new percentage. If an error */        2700
   28     /* occurs during the update, ROLLBACK the changes.                */        2800
   29 C  EXEC SQL WHENEVER SQLERROR GO TO UPDATEXERROR;                                2900
   30     EXEC SQL                                                                     3000
   31 D       UPDATE USER1/TEMPRACT                                                    3100
   32           SET EMPTIME = EMPTIME * (1+:PERCENTAGE)                                3200
   33           WHERE PROJNO LIKE :PROJID;                                             3300
   34                                                                                  3400
   35     /* Commit changes */                                                         3500
   36 E  EXEC SQL                                                                     3600
   37         COMMIT;                                                                  3700
   38     EXEC SQL WHENEVER SQLERROR GO TO REPORTXERROR;                               3800
   39                                                                                  3900
   40     /* Report the updated statistics for each employee assigned to the */        4000
   41     /* selected projects.                                             */        4100
   42                                                                                  4200
   43     /* Write out the header for Report 1 */                                      4300
   44     put file(sysprint) edit('UPDATED EMPLOYEE PROJECT ACCOUNT DATA')             4400
   45                             (col(1),a);                                          4500
   46     put file(sysprint)                                                           4600
   47         edit('EMPLOYEE','PROJECT','ACCOUNT','EMPLOYEE')                          4700
   48             (skip(2),col(1),a,col(11),a,col(20),a,col(29),a);                    4800
   49     put file(sysprint)                                                           4900
   50         edit('NUMBER','NUMBER','NUMBER','HOURS')                                 5000
   51             (skip,col(2),a,col(11),a,col(20),a,col(30),a,skip);                  5100
   52                                                                                  5200
   53     exec sql                                                                     5300
   54 F       declare c1 cursor for                                                    5400
   55           select *                                                               5500
   56           from user1/tempract                                                    5600
   57           where tempract.projno like :projid                                     5700
   58           order by empno;                                                        5800
   59     EXEC SQL                                                                     5900
   60 G       OPEN C1;                                                                 6000
                                                                                                    RSLS783-0
```

Figure D-2 (Part 2 of 6). Sample PL/I Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       PLIEX                    05/05/88 09:37:30   Page     3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
   61                                                                                          6100
   62      /* Fetch and write the rows to SYSPRINT */                                          6200
   63   H  EXEC SQL WHENEVER NOT FOUND GO TO DONE1;                                            6300
   64                                                                                          6400
   65      DO UNTIL (SQLCODE X= 0);                                                            6500
   66        EXEC SQL                                                                          6600
   67   I         FETCH C1 INTO :RPT1;                                                        6700
   68        PUT FILE(SYSPRINT)                                                                6800
   69            EDIT(RPT1.EMPNO,RPT1.PROJNO,RPT1.ACTNO,RPT1.EMPTIME)                          6900
   70                (SKIP,COL(2),A,COL(11),A,COL(21),F(5),COL(29),F(8,2));                    7000
   71      END;                                                                                7100
   72                                                                                          7200
   73    DONE1:                                                                                7300
   74   J  EXEC SQL                                                                            7400
   75           CLOSE C1;                                                                      7500
   76                                                                                          7600
   77      /* For each project selected, generate a report containing the     */              7700
   78      /* project number, project name, the old total of employee hours,  */              7800
   79      /* and the new total of employee hours for each project.           */              7900
   80                                                                                          8000
   81      /* Write out the header for Report 2 */                                             8100
   82      PUT FILE(SYSPRINT) EDIT('ACCUMULATED STATISTICS BY PROJECT')                        8200
   83                              (SKIP(3),COL(22),A);                                        8300
   84      PUT FILE(SYSPRINT)                                                                  8400
   85          EDIT('PROJECT','NUMBER OF','PREVIOUS','CURRENT')                                8500
   86              (SKIP(2),COL(1),A,COL(48),A,COL(59),A,COL(70),A);                           8600
   87      PUT FILE(SYSPRINT)                                                                  8700
   88          EDIT('NUMBER','PROJECT NAME','EMPLOYEES','HOURS','HOURS')                       8800
   89              (SKIP,COL(1),A,COL(10),A,COL(48),A,COL(60),A,COL(71),                       8900
   90               A,SKIP);                                                                   9000
   91                                                                                          9100
   92      EXEC SQL                                                                            9200
   93   K      DECLARE C2 CURSOR FOR                                                          9300
   94               SELECT TEMPRACT.PROJNO, PRNAME, COUNT(*),                                  9400
   95                  SUM(EMPTIME/(1.0+:PERCENTAGE)),SUM(EMPTIME)                             9500
   96               FROM USER1/TEMPRACT, USER1/TPROJ                                           9600
   97               WHERE TEMPRACT.PROJNO=TPROJ.PROJNO                                         9700
   98               GROUP BY TEMPRACT.PROJNO, PRNAME                                           9800
   99               HAVING TEMPRACT.PROJNO LIKE :PROJID                                        9900
  100               ORDER BY 1;                                                              10000
  101      EXEC SQL                                                                          10100
  102           OPEN C2;                                                                     10200
  103                                                                                        10300
  104      /* Fetch and write the rows to SYSPRINT */                                        10400
  105      EXEC SQL WHENEVER NOT FOUND GO TO DONE2;                                          10500
  106                                                                                        10600
  107      DO UNTIL (SQLCODE X= 0);                                                          10700
  108        EXEC SQL                                                                        10800
  109   L         FETCH C2 INTO :RPT2;                                                       10900
  110        PUT FILE(SYSPRINT)                                                              11000
  111            EDIT(RPT2.PROJNO,RPT2.PROJECTXNAME,EMPLOYEEXCOUNT,                          11100
  112                 OLDXTOTALXTIME,NEWXTOTALXTIME)                                         11200
  113                (SKIP,COL(1),A,COL(10),A,COL(50),F(4),COL(59),F(8,2),                   11300
  114                COL(69),F(8,2));                                                        11400
  115      END;                                                                              11500
  116                                                                                       11600
  117    DONE2:                                                                            11700
  118      EXEC SQL                                                                        11800
  119           CLOSE C2;                                                                  11900
  120      GO TO FINISHED;                                                                 12000
                                                                                                RSLS784-0
```

Figure D-2 (Part 3 of 6). Sample PL/I Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       PLIEX                    05/05/88 09:37:30   Page     4
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
  121                                                                                    12100
  122       /* Error occured while updating table.  Inform user and rollback   */        12200
  123       /* changes.                                                        */        12300
  124     UPDATEXERROR:                                                                12400
  125 M     EXEC SQL WHENEVER SQLERROR CONTINUE;                                       12500
  126       PUT FILE(SYSPRINT) EDIT('*** ERROR Occurred while updating table.'!!       12600
  127        '  SQLCODE=',SQLCODE)(A,F(5));                                            12700
  128       EXEC SQL                                                                   12800
  129 N          ROLLBACK;                                                           12900
  130       GO TO FINISHED;                                                          13000
  131                                                                                13100
  132       /* Error occured while generating reports.  Inform user and exit.  */     13200
  133     REPORTXERROR:                                                              13300
  134       PUT FILE(SYSPRINT) EDIT('*** ERROR Occurred while generating '!!         13400
  135        'reports.  SQLCODE=',SQLCODE)(A,F(5));                                  13500
  136        GO TO FINISHED;                                                         13600
  137                                                                                13700
  138       /* All done  */                                                          13800
  139     FINISHED:                                                                  13900
  140       CLOSE FILE(SYSPRINT);                                                    14000
  141       RETURN;                                                                  14100
  142                                                                                14200
  143     END PLIEX;                                                                 14300
                              * * * * *  E N D  O F  S O U R C E  * * * * *
                                                                                     RSLS785-0
```

Figure D-2 (Part 4 of 6). Sample PL/I Program Using SQL Statements

```
5728ST1 R01 M02 880901                  IBM SQL/400       PLIEX                    05/05/88 09:37:30   Page     5
                                        CROSS REFERENCE
Data Names                 Define    Reference
"ACTNO"                      56        SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"C1"                         54        CURSOR
                                       60 67 75
"C2"                         93        CURSOR
                                       102 109 119
"DEPTMGR"                    96        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"DEPTNO"                     96        CHARACTER(3) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"EMPNO"                      ****      COLUMN
                                       58
"EMPNO"                      56        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"EMPTIME"                    56        DECIMAL(5,2) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"EMPTIME"                    ****      COLUMN
                                       32 32 95 95
"ENDDATE"                    56        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"MAJPROJ"                    96        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PRENDATE"                   96        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PRNAME"                     ****      COLUMN
                                       94 98
"PRNAME"                     96        CHARACTER(36) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PROJNO"                     ****      COLUMN IN "USER1"."TEMPRACT"
                                       57
"PROJNO"                     56        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"PROJNO"                     ****      COLUMN IN "USER1"."TEMPRACT"
                                       94
"PROJNO"                     ****      COLUMN IN "USER1"."TEMPRACT"
                                       97
"PROJNO"                     ****      COLUMN IN "USER1"."TPROJ"
                                       97
"PROJNO"                     ****      COLUMN IN "USER1"."TEMPRACT"
                                       98
"PROJNO"                     ****      COLUMN IN "USER1"."TEMPRACT"
                                       99
"PROJNO"                     96        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PROJNO"                     ****      COLUMN
                                       33
"PRSTAFF"                    96        DECIMAL(5,2) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PRSTDATE"                   96        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"STARTDATE"                  56        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"TEMPRACT"                   ****      TABLE IN "USER1"
                                       31 56 57 94 96 97 98 99
"TPROJ"                      ****      TABLE IN "USER1"
                                       96 97
"USER1"                      ****      DATABASE
                                       31 56 96 96

                                                                                                    RSLS786-0
```

Figure D-2 (Part 5 of 6). Sample PL/I Program Using SQL Statements

```
5728ST1 R01 M02 880901                IBM SQL/400       PLIEX          05/05/88 09:37:30   Page     6
                                      CROSS REFERENCE
ACTNO                          11     SMALL INTEGER PRECISION(4,0) IN RPT1
DONE1                          ****   LABEL
                                      63
DONE2                          ****   LABEL
                                      105
EMPLOYEEXCOUNT                 17     SMALL INTEGER PRECISION(4,0) IN RPT2
EMPNO                          11     CHARACTER(6) IN RPT1
EMPTIME                        11     DECIMAL(5,2) IN RPT1
ENDDATE                        11     CHARACTER(6) IN RPT1
NEWXTOTALXTIME                 19     DECIMAL(8,2) IN RPT2
OLDXTOTALXTIME                 18     DECIMAL(8,2) IN RPT2
PERCENTAGE                     4      DECIMAL(5,2)
                                      32 95
PROJECTXNAME                   16     CHARACTER(36) IN RPT2
PROJID                         3      CHARACTER(3)
                                      33 57 99
PROJNO                         11     CHARACTER(6) IN RPT1
PROJNO                         15     CHARACTER(6) IN RPT2
REPORTXERROR                   ****   LABEL
                                      38
RPT1                           10     STRUCTURE
                                      67
RPT2                           14     STRUCTURE
                                      109
STARTDATE                      11     CHARACTER(6) IN RPT1
SYSPRINT                       7
UPDATEXERROR                   ****   LABEL
No errors found in source
  143 Source records processed
                      * * * * *  E N D  O F  L I S T I N G  * * * * *
```

RSLS787-0

Figure D-2 (Part 6 of 6). Sample PL/I Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400      RPGEX          05/09/88 15:35:23   Page   1
Source type..............RPG
Program name.............DEDRA/RPGEX
Source file..............*LIBL/QRPGSRC
Member...................*PGM
Options..................*SRC      *XREF
INCLUDE file.............*LIBL/*SRCFILE
Commit...................*CHG
Generation level.........0
Printer file.............*LIBL/DEDRA
Text.....................*SRCMBRTXT
Source member changed on 05/09/88  14:56:15
                                                                                    RSLS766-0
```

Figure D-3 (Part 1 of 7). Sample RPG Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       RPGEX                    05/09/88 15:35:23   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
    1       H                                                                                    100
    2       F*  File declaration for QPRINT                                                      200
    3       F*                                                                                   300
    4       FQPRINT  O   F     132           PRINTER                                           400
    5       I*                                                                                   500
    6       I* Structure for report 1.                                                           600
    7       I*                                                                                   700
    8   A   IRPT1      E DSTEMPRACT                                                              800
    9       I              STARTDATE                     STARDT                                 900
   10       I              ENDDATE                       ENDDT                                 1000
   11       I              EMPTIME                       EMPTIM                                1100
   12       I*                                                                                  1200
   13       I* Structure for report 2.                                                          1300
   14       I*                                                                                  1400
   15       IRPT2        DS                                                                     1500
   16       I                                        1   6 PRJNUM                               1600
   17       I                                        7  42 PNAME                                1700
   18       I                                    B  43  440EMPCNT                              1800
   19       I                                    P  45  4920LDTIM                              1900
   20       I                                    P  50  542NEWTIM                              2000
   21       I*                                                                                 2100
   22       I            DS                                                                    2200
   23       I                                        1   3 PROJID                              2300
   24   B   I                                    P   4   62PERCNT                             2400
   25       C*                                                                                 2500
   26       C                    Z-ADD.06       PERCNT                                         2600
   27       C                    MOVEL'MA%'     PROJID                                         2700
   28       C*                                                                                 2800
   29       C* Update the selected projects by the new percentage. If an                       2900
   30       C* error occurs during the update, ROLLBACK the changes.                          3000
   31       C*                                                                                 3100
   32   C   C/EXEC SQL WHENEVER SQLERROR GOTO UPDERR                                          3200
   33       C/END-EXEC                                                                        3300
   34       C*                                                                                3400
   35   D   C/EXEC SQL                                                                        3500
   36       C+ UPDATE USER1/TEMPRACT                                                          3600
   37       C+    SET EMPTIME = EMPTIME * (1+:PERCNT)                                         3700
   38       C+    WHERE PROJNO LIKE :PROJID                                                   3800
   39       C/END-EXEC                                                                        3900
   40       C*                                                                                4000
   41       C* Commit changes.                                                                4100
   42       C*                                                                                4200
   43   E   C/EXEC SQL COMMIT                                                                 4300
   44       C/END-EXEC                                                                        4400
   45       C*                                                                                4500
   46       C/EXEC SQL WHENEVER SQLERROR GO TO RPTERR                                         4600
   47       C/END-EXEC                                                                        4700
   48       C*                                                                                4800
   49       C* Report the updated statistics for each employee assigned to                     4900
   50       C* selected projects.                                                             5000
   51       C*                                                                               5100
   52       C* Write out the header for report 1.                                            5200
   53       C*                                                                              5300
   54       C                      EXCPTRECA                                                5400
   55   F   C/EXEC SQL declare c1 cursor for                                                 5500
   56       C+    select * from user1/tempract                                             5600
   57       C+    where tempract.projno like :projid                                       5700
   58       C+    order by empno                                                           5800
   59       C/END-EXEC                                                                     5900
   60       C*                                                                             6000
                                                                                                RSLS767-0
```

Figure D-3 (Part 2 of 7). Sample RPG Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       RPGEX          05/09/88 15:35:23   Page     3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
   61      C/EXEC SQL                                                                          6100
   62   G  C+  OPEN C1                                                                        6200
   63      C/END-EXEC                                                                          6300
   64      C*                                                                                  6400
   65      C* Fetch and write the rows to QPRINT.                                              6500
   66      C*                                                                                  6600
   67      C/EXEC SQL WHENEVER NOT FOUND GO TO DONE1                                           6700
   68   H  C/END-EXEC                                                                         6800
   69      C           SQLCOD    DOUNE0                                                        6900
   70      C/EXEC SQL                                                                          7000
   71   I  C+   FETCH C1 INTO :RPT1                                                           7100
   72      C/END-EXEC                                                                          7200
   73      C                     EXCPTRECB                                                     7300
   74      C                     END                                                           7400
   75      C           DONE1     TAG                                                           7500
   76   J  C/EXEC SQL                                                                         7600
   77      C+  CLOSE C1                                                                        7700
   78      C/END-EXEC                                                                          7800
   79      C*                                                                                  7900
   80      C* For each project selected, generate a report containing the                      8000
   81      C* project number, project name, the old total of employee hours,                   8100
   82      C* and the new total of employee hours.                                            8200
   83      C*                                                                                  8300
   84      C* Write out the header for report 2.                                               8400
   85      C*                                                                                  8500
   86      C                     EXCPTRECC                                                     8600
   87      C/EXEC SQL                                                                          8700
   88   K  C+  DECLARE C2 CURSOR FOR                                                          8800
   89      C+    SELECT TEMPRACT.PROJNO, PRNAME, COUNT(*),                                   8900
   90      C+       SUM(EMPTIME/(1.0+:PERCNT)),SUM(EMPTIME)                                  9000
   91      C+    FROM USER1/TEMPRACT, USER1/TPROJ                                            9100
   92      C+    WHERE TEMPRACT.PROJNO = TPROJ.PROJNO                                        9200
   93      C+    GROUP BY TEMPRACT.PROJNO, PRNAME                                            9300
   94      C+    HAVING TEMPRACT.PROJNO LIKE :PROJID                                         9400
   95      C+    ORDER BY 1                                                                  9500
   96      C/END-EXEC                                                                          9600
   97      C*                                                                                  9700
   98      C/EXEC SQL OPEN C2                                                                  9800
   99      C/END-EXEC                                                                          9900
  100      C*                                                                                 10000
  101      C* Fetch and write the rows to QPRINT.                                             10100
  102      C*                                                                                 10200
  103      C/EXEC SQL WHENEVER NOT FOUND GO TO DONE2                                          10300
  104      C/END-EXEC                                                                         10400
  105      C           SQLCOD    DOUNE0                                                       10500
  106      C/EXEC SQL                                                                         10600
  107   L  C+   FETCH C2 INTO :RPT2                                                          10700
  108      C/END-EXEC                                                                         10800
  109      C                     EXCPTRECD                                                    10900
  110      C                     END                                                          11000
  111      C           DONE2     TAG                                                          11100
  112      C/EXEC SQL CLOSE C2                                                                11200
  113      C/END-EXEC                                                                         11300
  114      C                     GOTO FINISH                                                  11400
  115      C*                                                                                 11500
  116      C* Error occured while updating table.  Inform user and rollback                   11600
  117      C* changes.                                                                        11700
  118      C*                                                                                 11800
  119      C           UPDERR    TAG                                                          11900
  120      C                     EXCPTRECE                                                    12000
                                                                                                    RSLS768-0
```

Figure D-3 (Part 3 of 7). Sample RPG Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       RPGEX          05/09/88 15:35:23   Page     4
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
  121     C/EXEC SQL WHENEVER SQLERROR CONTINUE                                               12100
  122   M C/END-EXEC                                                                         12200
  123     C*                                                                                  12300
  124   N C/EXEC SQL                                                                          12400
  125   N C+   ROLLBACK                                                                      12500
  126     C/END-EXEC                                                                          12600
  127     C                   GOTO FINISH                                                     12700
  128     C*                                                                                  12800
  129     C* Error occured while generating reports.  Inform user and exit.                   12900
  130     C*                                                                                  13000
  131     C           RPTERR  TAG                                                             13100
  132     C                   EXCPTRECF                                                       13200
  133     C*                                                                                  13300
  134     C* All done.                                                                        13400
  135     C*                                                                                  13500
  136     C           FINISH  TAG                                                             13600
  137     C                   SETON                     LR                                    13700
  138     OQPRINT  E 0201         RECA                                                        13800
  139     O                                   21 'UPDATED EMPLOYEE PROJ'                      13900
  140     O                                   37 'ECT ACCOUNT DATA'                           14000
  141     O        E 01           RECA                                                        14100
  142     O                                    8 'EMPLOYEE'                                   14200
  143     O                                   17 'PROJECT'                                    14300
  144     O                                   26 'ACCOUNT'                                    14400
  145     O                                   36 'EMPLOYEE'                                   14500
  146     O        E 02           RECA                                                        14600
  147     O                                    7 'NUMBER'                                     14700
  148     O                                   16 'NUMBER'                                     14800
  149     O                                   25 'NUMBER'                                     14900
  150     O                                   34 'HOURS'                                      15000
  151     O        E 01           RECB                                                        15100
  152     O                       EMPNO        7                                             15200
  153     O                       PROJNO      16                                             15300
  154     O                       ACTNO L     26                                             15400
  155     O                       EMPTIML     37                                             15500
  156     O        E 22           RECC                                                        15600
  157     O                                   42 'ACCUMULATED STATISTIC'                      15700
  158     O                                   54 'S BY PROJECT'                               15800
  159     O        E 01           RECC                                                        15900
  160     O                                    7 'PROJECT'                                    16000
  161     O                                   56 'NUMBER OF'                                  16100
  162     O                                   66 'PREVIOUS'                                   16200
  163     O                                   76 'CURRENT'                                    16300
  164     O        E 02           RECC                                                        16400
  165     O                                    6 'NUMBER'                                     16500
  166     O                                   21 'PROJECT NAME'                               16600
  167     O                                   56 'EMPLOYEES'                                  16700
  168     O                                   64 'HOURS'                                      16800
  169     O                                   75 'HOURS'                                      16900
  170     O        E 01           RECD                                                        17000
  171     O                       PRJNUM       6                                             17100
  172     O                       PNAME       45                                             17200
  173     O                       EMPCNTL     55                                             17300
  174     O                       OLDTIML     67                                             17400
  175     O                       NEWTIML     77                                             17500
  176     O        E 01           RECE                                                        17600
  177     O                                   28 '*** ERROR Occurred while'                   17700
  178     O                                   52 ' updating table. SQLCODE'                   17800
  179     O                                   53 '='                                          17900
  180     O                       SQLCODL     62                                             18000
                                                                                                    RSLS769-0
```

Figure D-3 (Part 4 of 7). Sample RPG Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       RPGEX                    05/09/88 15:35:23   Page     5
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
  181       0       E 01           RECF                                                      18100
  182       0                                 28 '*** ERROR Occurred while'                  18200
  183       0                                 52 ' generating reports. SQL'                  18300
  184       0                                 57 'CODE='                                     18400
  185       0                       SQLCODL   67                                             18500
                              * * * * *  E N D  O F  S O U R C E  * * * * *
```

RSLS770-0

Figure D-3 (Part 5 of 7). Sample RPG Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400       RPGEX                    05/09/88 15:35:23   Page     6
                                          CROSS REFERENCE
Data Names                      Define    Reference
"ACTNO"                           35        SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"C1"                              55        CURSOR
                                            61 70 76
"C2"                              87        CURSOR
                                            98 106 112
"DEPTMGR"                         87        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"DEPTNO"                          87        CHARACTER(3) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"EMPNO"                           35        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"EMPNO"                           ****      COLUMN
                                            55
"EMPTIME"                         ****      COLUMN
                                            35 35 87 87
"EMPTIME"                         35        DECIMAL(5,2) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"ENDDATE"                         35        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"MAJPROJ"                         87        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PRENDATE"                        87        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PRNAME"                          ****      COLUMN
                                            87 87
"PRNAME"                          87        CHARACTER(36) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PROJNO"                          ****      COLUMN
                                            35
"PROJNO"                          35        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"PROJNO"                          ****      COLUMN IN "USER1"."TEMPRACT"
                                            55
"PROJNO"                          ****      COLUMN IN "USER1"."TEMPRACT"
                                            87
"PROJNO"                          ****      COLUMN IN "USER1"."TEMPRACT"
                                            87
"PROJNO"                          ****      COLUMN IN "USER1"."TPROJ"
                                            87
"PROJNO"                          ****      COLUMN IN "USER1"."TEMPRACT"
                                            87
"PROJNO"                          ****      COLUMN IN "USER1"."TEMPRACT"
                                            87
"PROJNO"                          87        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PRSTAFF"                         87        DECIMAL(5,2) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"PRSTDATE"                        87        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TPROJ"
"STARTDATE"                       35        CHARACTER(6) COLUMN (NOT NULL) IN "USER1"."TEMPRACT"
"TEMPRACT"                        ****      TABLE IN "USER1"
                                            35 55 55 87 87 87 87 87
"TPROJ"                           ****      TABLE IN "USER1"
                                            87 87
"USER1"                           ****      DATABASE
                                            35 55 87 87
```

RSLS771-0

Figure D-3 (Part 6 of 7). Sample RPG Program Using SQL Statements

```
5728ST1 R01 M02 880901                    IBM SQL/400      RPGEX          05/09/88 15:35:23   Page     7
                                          CROSS REFERENCE
ACTNO                          8          SMALL INTEGER PRECISION(4,0)
DONE1                          75         LABEL
                                          67
DONE2                          111        LABEL
                                          103
EMPCNT                         18         SMALL INTEGER PRECISION(4,0) IN RPT2
EMPNO                          8          CHARACTER(6)
EMPTIM                         8          DECIMAL(5,2)
ENDDT                          8          CHARACTER(6)
FINISH                         136        LABEL
NEWTIM                         20         DECIMAL(9,2) IN RPT2
OLDTIM                         19         DECIMAL(9,2) IN RPT2
PERCNT                         24         DECIMAL(5,2)
                                          35 87
PNAME                          17         CHARACTER(36) IN RPT2
PRJNUM                         16         CHARACTER(6) IN RPT2
PROJID                         23         CHARACTER(3)
                                          35 55 87
PROJNO                         8          CHARACTER(6)
RPTERR                         131        LABEL
                                          46
RPT1                           8          CHARACTER(29)
                                          70
RPT2                           15         STRUCTURE
                                          106
STARDT                         8          CHARACTER(6)
UPDERR                         119        LABEL
No errors found in source
  185 Source records processed
                          ***** E N D  O F  L I S T I N G *****
                                                                                          RSLS772-0
```

Figure D-3 (Part 7 of 7). Sample RPG Program Using SQL Statements

Report Produced by Sample Programs

The following report is produced by each of the preceding sample programs.

```
UPDATED EMPLOYEE PROJECT ACCOUNT DATA

EMPLOYEE  PROJECT  ACCOUNT  EMPLOYEE
 NUMBER   NUMBER   NUMBER    HOURS

 000060   MA2100       10    530.00
 000110   MA2100       20    424.00
 000160   MA2100       20    530.00
 000170   MA2100       20    530.00
 000170   MA2112       70    106.00
 000180   MA2100       20    689.00
 000180   MA2113       70    424.00
 000190   MA2112       70    106.00
 000210   MA2113       80    530.00
 000220   MA2112       50    954.00
 000230   MA2113       70    318.00

                ACCUMULATED STATISTICS BY PROJECT

PROJECT                                  NUMBER OF  PREVIOUS   CURRENT
NUMBER   PROJECT NAME                    EMPLOYEES   HOURS     HOURS

MA2100   MFG AUTOMATION                         5    2550.00   2703.00
MA2112   ROBOT DESIGN                           3    1100.00   1166.00
MA2113   PROD CONTROL PROG                      3    1200.00   1272.00
```

RSLS765-0

Glossary

access path. The path used to locate data specified in SQL statements. An access path can be either indexed or sequential, or a combination of both.

access plan. The control structure produced during compile time that is used to process SQL statements encountered when the program is run.

ANSI. American National Standards Institute

application. A program or set of programs that perform a task; for example, a payroll application.

attribute. In data base design, a characteristic of an entity; for example, the telephone number of an employee is one of that employee's attributes.

authorization ID. A user profile. A name identifying a user to whom privileges can be granted.

automatic bind. When an application program is being run and the access plan is not valid, binding takes place automatically; that is, without a user issuing a Bind command.

binary. An SQL data type indicating that the data is a binary number with a precision of 15 (halfword) or 31 (fullword) bits.

bind. The process by which the output from the SQL precompiler is converted to a usable structure called an access plan. This process is the one during which access paths to the data are selected and some authorization checking is performed. There are two types of bind used by SQL/400: automatic and dynamic (see *automatic bind* and *dynamic bind*).

catalog. Tables, maintained by the data base utility, that contain descriptions of objects, such as tables, views, and indexes.

catalog views. A set of views containing information about the objects in a data base, such as tables, views, indexes, and column definitions.

character string. A sequence of bytes or characters associated with a single-byte character set.

clause. A distinct part of a statement in the language structure, such as a SELECT clause or a WHERE clause.

column. The vertical part of a table. A column has a name and a particular data type (for example, character, decimal, or integer).

column function. A process that calculates a value from a set of values and expresses it as a function name followed by an argument enclosed in parentheses.

commit. The process that data changed by one application or user to be used by other applications or users. When a commit operation occurs, the locks are released to allow other applications to use the changed data.

commit point. The point in time when data is considered to be consistent.

comparison operator. A symbol (such as =, >, <) used to specify a relationship between two values.

concurrency. The shared use of resources by multiple interactive users or application programs at the same time.

correlation name. An identifier that designates a table, a view, or an individual row of a table or view within a single SQL statement. The name can be defined in any FROM clause or in the first clause of an UPDATE or DELETE statement.

cursor. A named control structure used by an application program to point to a row of data. The position of the row is within a table or view, and the cursor is used to interactively select rows from the columns.

data base. A set of objects created by SQL/400 that contains tables, views, indexes, and other system objects (such as a program) created by the user. An SQL data base consists of a library; a data dictionary that contains description and information for all tables, views, indexes, and files created into the library; an SQL catalog; and a journal and journal receiver that are used to journal changes to all tables created into the data base.

data type. An attribute of columns, constants, and host variables.

DBCS. See *double-byte character set (DBCS)*.

default value. A predetermined value, attribute, or option that is assumed when no other value is explicitly specified. For example, the value of a column is a nonnull value determined by the data type of the column.

delimited identifier. A sequence of one or more characters of the standard character set enclosed within SQL escape characters used to form a name.

delimiter token. A string constant, a delimited identifier, a symbol (for example, ||, /, *, +, or –), or other

special characters (for example, period, comma, parentheses).

double byte character set (DBCS). A set of characters used by national languages, such as Japanese and Chinese, that have more symbols than can be represented by the 256 single-byte EBCDIC positions. Each character is 2 bytes in length, and therefore requires special hardware to be displayed or printed. Contrast with single-byte character set.

dynamic bind. When SQL statements are entered interactively, binding is done dynamically (that is, as the SQL statements are entered).

dynamic SQL. SQL statements that are prepared and processed within a program while the program is running. The SQL source statements are contained in host-language variables rather than being coded directly into the application program. The SQL statement might change several times while the program is running.

EBCDIC. See *extended binary coded decimal interchange code (EBCDIC)*.

embedded SQL. SQL statements that are embedded within a program and are prepared during the program preparation process before the program is run. After it is prepared, the statement itself does not change, although values of host variables specified within the statement might change.

escape character. The symbol used to enclose a delimited identifier. This symbol is the quotation mark ("), except in COBOL programs where the symbol can be assigned by the user as either a quotation mark or an apostrophe.

expression. An operand, or a collection of operators and operands, that yields a single value.

extended binary coded decimal interchange code (EBCDIC). A coded character set of 256 8-bit characters.

fixed-length string. A character string whose length is specified and cannot be changed. Contrast with varying-length string.

fullword binary. A binary number with a precision of 31 bits. See also *integer*.

full select. That form of the SELECT statement that includes ORDER BY or UNION operators.

function. A column function or a scalar function.

halfword binary. A binary number with a precision of 15 bits.

host language. Any programming language, such as COBOL, PL/I, and RPG, in which you can embed SQL statements.

host structure. In an application program, a structure referred to by embedded SQL statements. In RPG, this is called a *data structure*; in PL/I, this is known as a *structure*; in COBOL, this is called a *group item*.

host variable. In an application program, a variable referred to by embedded SQL statements. In RPG, this is called a *field name*; in PL/I, this is known as a *variable*; in COBOL, this is called a *data item*.

identifier. See *delimited identifier* and *ordinary identifier*.

index. A set of pointers that are logically arranged by the values of a key. Indexes provide quick access to data and can enforce uniqueness on the rows in a table.

index key. The set of columns in a table used to determine the order of indexed entries.

indicator variable. A variable used to represent the null value in an application program. For example, if the value for the results column is null, SQL puts a negative value in the indicator variable.

integer. An SQL data type indicating that the data is a binary number with a precision of 31 bits.

join. A relational operation that allows retrieval of data from two or more tables based on matching column values.

key. A column or an ordered collection of columns identified in the description of an index.

keyword. A name that identifies a parameter used in an SQL statement or SQL precompiler command. See also *parameter*.

lock. The process by which integrity of data is ensured. The prevention of concurrent users from accessing inconsistent data.

long string. A string whose actual length, or a varying-length string whose maximum length, is greater than 254 bytes or 127 double-byte characters.

menu. A displayed list of available, logically grouped functions for selection by the operator.

mixed data string. A character string that can contain both single-byte and double-byte characters.

null. A special value that indicates the absence of information.

object. Anything that can be created or manipulated with SQL statements, such as data bases, tables, views, or indexes.

ordinary identifier. A letter followed by zero or more characters, each of which is a letter ($, @, #, a-z, and A-Z), a number, or the underscore character used to form a name. An ordinary identifier must not be identical to a reserved word.

ordinary token. A numeric constant, and ordinary identifier, a host variable, or a keyword.

page. A unit of storage equal to 512 bytes.

parameter. The *keywords* and *values* that further define SQL precompiler commands and SQL statements.

plan. See *access plan.*

precompile. A processing of programs containing SQL statements that takes place before a compile. SQL statements are replaced with statements that will be recognized by the host language compiler. The output from this precompile includes source code that can be submitted to the compiler and used in the bind process.

predicate. An element of a search value that expresses or implies a comparison operation.

prepared SQL statement. A named object that is the form of an SQL statement that was processed by the PREPARE statement.

privilege. A capability given to a user by the processing of a GRANT statement.

rebind. The creation a new access plan for a program that was previously bound. If, for example, you add an index for a table that is used by your application program, SQL/400 may automatically bind the application again to take advantage of that index.

real table. A physical file or a table created by SQL.

recovery. The process of rebuilding data bases after a system failure.

relational data base. A data structure perceived by its users as a collection of tables.

results column. The set of columns that SQL selects for an application program.

results table. The set of rows that SQL selects for an application program. The program uses a cursor to retrieve the rows one by one into a host structure or a set of host variables.

rollback. The process of restoring data changed by an application to the state at its last commit point.

row. The horizontal part of a table. A row consists of a sequence of values, one for each column of the table.

SBCS. See *single-byte character set (SBCS).*

scalar function. An operation that produces a single value from another value and expresses it in the form of a function name followed by a list of arguments enclosed in parentheses.

search condition. A criterion for selecting rows from a table. A search condition consists of one or more predicates.

short string. A string whose actual length, or a varying-length string whose maximum length, is 254 bytes.

single-byte character set (SBCS). A character set in which each character is represented by a one-byte code.

small integer. An SQL data type indicating that the data is a binary number with a precision of 15 bits.

special register. A storage area whose primary use is to store information produced in conjunction with the use of specific SQL functions. The SQL/400 special register is (named) USER.

SQL. See *Structured Query Language.*

SQLCA. See *SQL communication area (SQLCA).*

SQLDA. See *SQL descriptor area (SQLDA).*

SQL communication area (SQLCA). A collection of variables that are used by SQL to provide an application program with information about the processing of SQL statements within the program.

SQL descriptor area (SQLDA). A collection of variables that are used in the processing of certain SQL statements. The SQLDA is intended for dynamic SQL programs.

static SQL. SQL statements that are embedded within a program, and are prepared during the program preparation process before the program is run. After being prepared, the statement itself does not change (although values of host variables specified by the statement might change).

string. A character string.

string delimiter. A symbol used to enclose an SQL string constant. This symbol is the apostrophe ('), except in COBOL applications, in which case the symbol (either an apostrophe or a quotation mark) may be assigned by the user.

Structured Query Language (SQL). A language that can be used within host programming languages or interactively to access data and to control access to resources.

subselect. That form of the SELECT statement that does not include ORDER BY or UNION operators.

table. A named data object consisting of a specific number of columns and some number of unordered rows.

token. See *delimited token* and *ordinary token.*

union. An SQL operation that combines the results of two select statements. Union is often used to merge lists of values obtained from several tables.

unique index. An index that assures that no identical key values are stored in a table.

unit of recovery. A sequence of operations within a unit of work between two commit points.

unlock. To release an object or system resource that was previously locked and return it to general availability.

user profile. An object with a unique name that contains the user's password, the list of special authorities assigned to a user, and the objects the user owns. See also *authorization ID.*

value. Smallest unit of data manipulated in SQL.

varying-length string. A character string whose length is not fixed, but variable within limits. Contrast with *fixed-length string.*

view. An alternative representation of data from one or more tables. A view can include all or some of the columns contained in the table or tables on which it is defined.

Index

Special Characters

***ALL 9-15**
***APOST 9-14**
***APOSTSQL 9-14**
***CHG 9-15**
***COMMA 9-14**
***CURLIB 9-12, 9-14, 9-15**
***GEN 9-13**
***LIBL 9-14, 9-15**
***NOGEN 9-13**
***NONE 9-15**
***NOSOURCE 9-13**
***NOSRC 9-13**
***NOXREF 9-13**
***PERIOD 9-14**
***PGM (program)**
description 9-13
***QUOTE 9-14**
***QUOTESQL 9-13**
***SAVLIB CL command 11-5**
***SAVOBJ CL command 11-5**
***SOURCE 9-1, 9-13**
***SQL 9-14**
***SRC 9-13**
***SRCFILE 9-15**
***SRCFILE (source file) 9-14**
***SRCMBRTXT 9-16**
***STRJRNAP CL command 11-5**
***SYS 9-14**
***SYSVAL 9-14**
***XREF 9-1, 9-13**
%INCLUDE directive 7-8
example 7-11
prefix 7-9
structure ending 7-10
structures 7-9
using prefixes 7-9

A

abbreviations
COBOL 5-7
PL/I 7-7
access plans 9-17
adding to the end of a table 13-9
address variables used in dynamic SQL 8-1
advanced coding techniques
complex search conditions 3-26
insert multiple rows into a table 3-34
notes on 3-34
join 3-29
selecting data from more than one table 3-29
allocating storage
SQLDA, example 8-10
allowable declarations for host variables
COBOL 5-5
PL/I 7-5
RPG 6-5
analyzing error and warning messages
output from the precompiler 9-3
AND keyword 3-26
multiple search condition 3-27
application plans
See access plans
application requirements
COBOL
communication area (SQLCA) 5-1
PL/I
communication area (SQLCA) 7-1
RPG
communication area 6-1
arithmetic expression error 3-6
arranging rows 3-14
arrays
COBOL 5-7
ASP (auxiliary storage pool) 11-3
assignment rules 4-2
character string assignment 4-3
COBOL 5-5
numeric assignment 4-3
PL/I 7-5
RPG 6-6
asterisk (select all columns) 3-7
atomic operations
data definition statements 11-3
data integrity 11-3
audience 10-3
authorization
testing 12-1
to create test tables
GRANT statement example 12-2
authorization ID 11-1
auxiliary storage pool (ASP) 11-3

B

basic SQL statements and clauses 3-1
BETWEEN clause
multiple search condition 3-27
BETWEEN keyword 3-26
binding 9-17
access plans 9-17
blocking considerations 9-18
BOTH, PL/I element type 7-8

C

calculation section (RPG) 6-2
catalog description
catalog views 1-6
catalog views 1-6
description of 1-6
SYSCOLUMNS 1-6
SYSINDEXES 1-8
SYSKEYS 1-8
SYSTABLES 1-9
SYSVIEWDEP 1-9
SYSVIEWS 1-10
catalogs 1-5, 2-1
changing information in a table 2-6
description 1-5
get information from 2-4
updating 2-6
using the catalog in data base design 2-11
working with 2-1, 2-4, 2-11
changing
indexes 1-5, 2-11
changing data 3-3
changing information in a table 2-6
changing the table definition 13-10
character data type 1-4
character string assignment 4-3
clauses 3-1
AND 3-27
NOT 3-27
OR 3-27
WHERE 3-27
CLOSE CURSOR statement 3-23, 3-25
COBOL programs 5-1
abbreviations 5-7
allowable declarations for host variables 5-5, 5-7
application requirements 5-1
arrays 5-7
assignment rules 5-5
coding requirements 5-2
comments 5-3
communication area (SQLCA) 5-1
continuation for SQL statements 5-3
data description entry
data types 5-7
descriptor area (SQLDA) 5-3
dynamic SQL 5-3
error and warning messages 9-17
during a compile 9-17
host structures 5-3
host variables 5-3
indicator variables 5-8, 7-7
OCCURS clause 5-7
overflow 5-7
procedure division 5-2
requirements for host variables 5-4
reserved words 5-3
return code handling 5-8
sample program with SQL statements D-3
COBOL programs *(continued)*
SQL delimiters 5-2
SQL precompiler command example 9-16
SQL syntax in 5-2
structures 5-8, 7-7
truncation 5-7
using SQL statements 5-1
value item 5-7
varying-length character string 5-7
length item 5-7
COBOL source file parameter (QLBLSRC) 9-13
coding examples D-1
SQL statements used in COBOL D-3
SQL statements used in PL/I D-11
SQL statements used in RPG D-17
coding required with host variable
naming conventions 7-4
coding requirements 7-4
COBOL 5-2, 5-3
comments 5-3
dynamic SQL 5-3
host structures 5-3
host variables 5-3, 5-4
including code 5-2
line continuation 5-3
margins 5-3
reserved words 5-3
sequence numbers 5-3
SQL statement delimiters 5-2
PL/I 7-1, 7-2
comments 7-2
dynamic SQL 7-1
host structures 7-3
host variables 7-3, 7-4
line continuation 7-2
reserved words 7-3
SQL statement delimiters 7-2
RPG 6-2, 6-3
comments 6-2
dynamic SQL 6-3
host variables 6-3
line continuation 6-2
reserved words 6-3
SQL statement delimiters 6-2
coding requirements, RPG
sequence numbers 6-3
coding SQL statements in COBOL 5-1
application requirements 5-1
host structures 5-3
return code handling 5-8
rules for host variables 5-4
sample program D-3
coding SQL statements in PL/I 7-1
application requirements 7-1
host structures 7-9
return code handling 7-11
rules for host variable 7-4
sample program D-11

coding SQL statements in RPG 6-1
application requirements 6-1
host structures 6-4
return code handling 6-7
rules for host variables 6-5
sample program D-17
coding techniques 3-1
collating rows 3-14
columns 1-3
combining 3-29
creating 2-2
data types 1-4
defining column headings 2-2
description 1-3
FOR UPDATE OF clause 3-24
insert default values into columns 3-35
inserting information 2-2
join 3-29
length and data type rules 3-18
numeric, rules for 3-18
selecting all the columns 3-7
specifying the columns you want 3-7
updating in views 2-10
combining information from more than one table 2-5
combining SELECT statements 3-16
COMMA
used with PL/I 7-9
commands 9-8, 9-9
CRTSQLCBL 9-9
CRTSQLPLI 9-10
CRTSQLRPG 9-11
for SQL COBOL precompiler 9-9
for SQL PL/I precompiler (CRTSQLPLI) 9-10
for SQL RPG precompiler (CRTSQLRPG) 9-11
help for CL commands v
keywords 9-12
parameter definitions 9-12
SQL precompiler 9-8, 9-12
CRTSQLCBL 9-8
CRTSQLPLI 9-8
CRTSQLRPG 9-8
values 9-12
COMMENT ON, using
getting comments 2-13
comments
COBOL 5-3
PL/I 7-2
RPG 6-2
COMMIT 9-15
interactive keyword 10-15
COMMIT statement 2-3, 2-6, 3-4, 11-4
HOLD value 11-4
commitment control 11-4
changing 10-4
common data base problems
techniques for solving 13-7
communication area (SQLCA)
COBOL 5-1
communication area (SQLCA) *(continued)*
PL/I 7-1
RPG 6-1
comparison operators 3-9
compiled application program objects 1-10
data dictionary 1-3
program 1-11
temporary source file member 1-11
user source file member 1-10
compiling
application programs 9-16
COBOL 9-17
error and warning messages 9-17
PL/I 9-17
error and warning messages 9-17
RPG 9-17
completing a unit of recovery 3-25
complex search conditions 3-26
keywords for use in search conditions 3-26
multiple search condition 3-27
WHERE 3-27
concatenate rows 3-31
concatenating two sets of selected rows 3-16
concepts 1-1
concepts, SQL with host languages 4-1
assignment rules 4-2
host structures 4-4
indicator variables 4-4
return code, handling 4-6
using host variables 4-1
concurrency, data integrity 11-2
continuation for SQL statements
COBOL 5-3
PL/I 7-2
RPG 6-2
conversion error 3-6
correlation names 2-5, 3-33
CREATE DATABASE statement 2-1
create SQL statements 9-9, 9-10, 9-11
for COBOL precompiler (CRTSQLCBL) 9-9
for PL/I precompiler 9-10
for RPG precompiler 9-11
CREATE TABLE statement 2-1, 2-2
CREATE VIEW statement 2-8
creating 2-2
data bases 2-1
indexes 2-11
rows 2-2
tables 2-1, 2-2
TDEPT (department table) 2-2
USER1 data base 2-1
creating a view 2-8
combining data from more than one table 2-9
on a single table 2-8
creating a view combining data from more than one table 2-9
creating a view on a single table 2-8

CRTDTADCT CL command 2-1
CRTDUPOBJ CL command 12-2
CRTLIB CL command 2-1
CRTSQLCBL 9-9
parameter description 9-12
syntax description 9-9
CRTSQLCBL command example 9-16
CRTSQLPLI 9-10
parameter description 9-12
syntax description 9-10
CRTSQLRPG 9-11
parameter description 9-12
syntax description 9-11
CURSOR statement 3-25
cursors 3-20
CLOSE CURSOR 3-23
completion of unit of recovery 3-25
defining a cursor 3-22
end of data 3-23
functions 3-20
OPEN CURSOR 3-23
OPEN CURSOR statement with DESCRIPTOR clause 8-16
selecting a set of rows 3-20
setting up a cursor 3-22
using 3-20, 8-14
using parameter markers 8-15

D

damage tolerance 11-5
data bases 1-3
changing list types 10-4
creating 2-1
description 1-3
techniques for solving problems 13-7
terminology 1-2
working with 2-1
data definition statements 11-3
data description entry
COBOL 5-7
data dictionary 1-3
data integrity 11-1, 11-2
atomic operations 11-3
commitment control 11-4
concurrency 11-2
damage tolerance 11-5
data definition statements 11-3
index recovery 11-5
journaling 11-3
save/restore 11-5
data items 5-3
COBOL 5-3
PL/I 7-3
RPG 6-3
data protection 11-1
data integrity 11-2
security 11-1
data protection *(continued)*
See also security.
data security 11-1
data types 1-4
character 1-4
COBOL 5-7
decimal 1-4
float 1-4
for COBOL applications 5-5
for PL/I applications 7-5
for RPG applications 6-5
integer 1-4
numeric 1-4
PL/I 7-7
real 1-4
SMALLINT (small integer) 1-4
DBCS 9-2, 10-14
entering in interactive SQL 10-14
use as precompiler input 9-2
DDM considerations 9-19
deadlock detection 11-2
debugging your program 12-2
decimal data type 1-4
declarations for host variables
COBOL 5-5, 5-7
PL/I 7-5
RPG 6-5
DECLARE CURSOR statement 3-5
defining 3-22
FOR UPDATE OF clause 3-24
selecting several rows at a time 3-5
setting up a cursor 3-22
declare restrictions
PL/I 7-7
defining table name and column headings 2-2
DELETE statement 2-7, 3-4
with WHERE clause 2-7
delete the current row 3-24
DELETE WHERE CURRENT OF statement 3-24
deleting information in a table 2-7
delimiters
for SQL statements in COBOL 5-2
SQL statements in PL/I 7-2
SQL statements in RPG 6-2
department table (TDEPT)
creating 2-2
DESCRIBE statement
using with dynamic SQL 8-10
descriptor area (SQLDA)
COBOL 5-3
description 8-7
dynamic SELECT statement format 8-7
fixed-list SELECT statement 8-5
format description 8-7, 8-8, 8-9
SQLD 8-8
SQLDABC 8-7
SQLDAID 8-7
SQLDATA 8-8
SQLDIND 8-9

descriptor area (SQLDA) *(continued)*
format description *(continued)*
SQLLEN 8-8
SQLN 8-7
SQLNAME 8-9
SQLRES 8-8
SQLTYPE 8-8
SQLVAR 8-8
PL/I 7-1
RPG 6-3
varying-list SELECT statement 8-5, 8-6
DESCRIPTOR clause with OPEN CURSOR 8-16
designing
test data structure 12-1
dictionary, data 1-3
displaying SQLCODE descriptions C-1
displays 10-3
flow diagram for interactive SQL 10-8
using interactive displays 10-3
DISTINCT keyword
using 3-25
double-byte character set (DBCS) 9-2
entering in interactive SQL 10-14
use as precompiler input 9-2
duplicate rows, eliminating 3-25
dynamic SQL 8-1
allocating storage 8-10
applications 8-1, 8-3
building and running statements 8-3
in COBOL programs 5-3
in PL/I programs 7-1
in RPG programs 6-3
processing 8-3
processing SELECT statements 8-3, 8-5
replacing parameter markers with host variables 8-16
requirements for using parameter markers 8-16
run-time overhead 8-1
statements 8-2
types of statements 8-3
use of address variables 8-1
using a cursor 8-14
using fixed-list 8-5
using OPEN CURSOR with DESCRIPTOR clause 8-16
using parameter markers 8-15
using the DESCRIBE statement 8-10
using the EXECUTE statement 8-4
using the PREPARE statement 8-4
using varying-list 8-5
valid SQL statements 8-1

E

element descriptions, PL/I 7-8, 7-9
COMMA 7-9
element type 7-8
BOTH 7-8
INDICATORS 7-8
INPUT 7-8
KEY 7-8
OUTPUT 7-8
RECORD 7-8
file name 7-8
format name 7-8
prefix name 7-9
element type
used with PL/I 7-8
BOTH 7-8
INDICATORS 7-8
INPUT 7-8
KEY 7-8
OUTPUT 7-8
RECORD 7-8
eliminating duplicate rows 3-25
end of data and cursors 3-23
END-EXEC 6-2
error and warning messages
during a COBOL compile 9-17
during a PL/I compile 9-17
output from the precompiler 9-3
error codes
handling by COBOL 5-8
handling by PL/I 7-11
handling for RPG 6-7
establishing a test environment 12-1
establishing position at the end of a table 13-9
examples iii
create SQL COBOL command 9-16
rules for SQL statements iii
using a cursor 3-20
exception condition handling 4-7, 6-7
by RPG 6-7
EXEC SQL 6-2
EXECUTE IMMEDIATE statement
use in PL/I 7-2
EXECUTE statement
using dynamically 8-4
exit interactive SQL 10-6
expressions 3-8
definition 3-8
external descriptions
RPG 6-6
external file descriptions
PL/I 7-8

F

failed session
recovery 10-7
FETCH statement 13-7
end of data 3-23
retrieving a row from a set of rows 3-23
file descriptions, external
PL/I 7-8
file name
used with PL/I 7-8
fixed-list
used with dynamic SQL 8-5
float data type 1-4
floating-point number 1-4, 4-3
FOR UPDATE OF clause 3-24
format name
used with PL/I 7-8
fullword binary interger 4-3
function
SQL precompiler 9-1
functional description 10-3
exiting interactive SQL 10-6
interactive SQL display flow diagram 10-8
list selection function 10-1, 10-5, 10-11
session history 10-4
session services 10-1, 10-4
statement entry 10-1, 10-3
supported SQL statements 10-8

G

generation level (GENLVL) 9-15
GENLVL 9-15
getting catalog information
about a column 2-11
about a table 2-11
about indexes 2-12
about views 2-12
getting comments 2-13
getting data from more than one table 3-29
getting information
from a single table 2-4
from more than one table 2-5
getting started using interactive SQL 10-3
GO TO label 4-8
GRANT statement 11-1, 12-2
example 12-2
GROUP BY clause 3-11
group of rows 3-12
grouping the rows you select 3-11
GRTOBJAUT (grant object authority) CL command 11-1
guidelines for using SQL 13-1
selecting data from multiple tables, improving performance 13-5
statements 13-1

H

halfword binary integer 4-3
HAVING clause 3-12
help
for CL commands, online v
for displays v
for SQL precompiler commands, online v
online 10-6
HOLD value 11-4
host languages
using SQL 4-1
host structures 4-4, 5-3
COBOL 5-3
definition 4-1
in a PL/I program 4-4
PL/I 7-3, 7-9
RPG 6-4
host variables 4-1
See also data items
address variables used in dynamic SQL 8-1
assignment rules for COBOL 5-5
assignment rules for PL/I 7-5
assignment rules for RPG 6-6
COBOL 5-3
COBOL declarations 5-5
input 8-15
INTO clause 4-1
other clauses 4-2
PL/I 7-3
PL/I declarations 7-5
replacing parameter markers with 8-16
requirements 5-4, 7-4
COBOL 5-4
PL/I 7-4
requirements for RPG 6-5
RPG 6-3
RPG declarations 6-5
RPG names 6-1
SELECT clause 4-2
using parameter markers 8-15
WHERE clause 4-1
how to
use SQL with host languages 4-1

I

IDDU (interactive data definition utility) 1-3
IN clause
multiple search condition 3-27
IN keyword 3-26
INCFILE 9-14
INCFILE parameter 9-2
include file (INCFILE) 9-14
INCLUDE SQLCA statement
COBOL 5-1
PL/I 7-1
RPG 6-1

INCLUDE statement
INCFILE parameter 9-2
use as precompiler input 9-2
including code in a COBOL program 5-2
index recovery 11-5
index search v
index, SQL
conditions with out an index 13-3
using effectively 13-3
indexes 1-5, 2-1
creating 2-11
working with 2-1, 2-11
indicator variables 4-4
COBOL 5-8
PL/I 7-7
RPG 6-6
INDICATORS, PL/I element type 7-8
input data
testing 12-1
input to precompiler 9-2
INPUT, PL/I element type 7-8
insert default values into columns 3-35
insert multiple rows into a table
insert default values into columns 3-35
INSERT statement 2-2, 3-1, B-3
blocking considerations 9-19
using B-3
VALUES clause 3-1
inserting
multiple rows into a table 3-34
rows 2-2
inserting information
columns 2-2
into tables 2-2
rows 2-2
inserting information into tables B-3
inserting multiple rows into a table
notes on 3-34
INTEGER
integer data type 1-4
integrity using views 11-1
integrity, data 11-1
See also security.
interactive SQL
basic functions 10-1
list selection 10-1
session services 10-1, 10-4
statement entry 10-1
statement entry function 10-1, 10-3
display flow diagram 10-8
entering DBCS data 10-14
functional description 10-3
getting started 10-3
help, online 10-6
overview 10-1
STRSQL command 10-15
supported SQL statements 10-8
tips on using interactive SQL 10-11
interactive SQL *(continued)*
used for testing SQL statements 10-14
using 10-3
interactive SQL statements 8-2
INTO clause 2-2, 3-5, B-3
used with host variables 4-1
using with INSERT 2-2, B-3
introduction
concepts 1-1
SQL objects description 1-3

J

join 3-29, 3-33
definition 2-5, 2-9
example 2-9
examples 2-5
notes on the technique 3-33
tables 3-29, 3-31
uses for 3-32
WHERE clause 3-31
joining tables 13-5
journal receiver 11-3
journaling 11-3
*STRJRNAP CL command 11-5
data integrity 11-3
index function 11-5
journals and journal receivers 1-3

K

keeping a copy of the data 13-7
keeping duplicates 3-19
KEY, PL/I element type 7-8
keywords
AND 3-26
BETWEEN 3-26
COMMIT 9-15, 10-15
DISTINCT 3-25
for use in search conditions 3-26
GENLVL 9-15
INCFILE (include file) 9-14
LIBOPT 10-16
LIKE 3-26
LSTTYP 10-16
MARGINS 9-15
NAMING 10-15
NOT keyword 3-10
OPTION 9-13
PGM (program) 9-12
PROCESS 10-15
PRTFILE 9-15
REFRESH 10-16
source file 9-13
source file name 9-13
SRCFILE 9-13
SRCMBR (source member) 9-13
TEXT 9-16
UNION 3-17

keywords *(continued)*
UNION ALL 3-19

L

LABEL ON statement 2-2
languages
using SQL with host 4-1
LCKLVL parameter 11-4
left right 9-15
length and data type rules 3-18
LIBOPT
interactive keyword 10-16
library
changing list types 10-4
library name 9-12, 9-14, 9-15
LIKE keyword 3-26
multiple search condition 3-27
line continuation
COBOL 5-3
PL/I 7-2
RPG 6-2
list selection
data base types 10-4
library types 10-4
qualifying names 10-5
list selection function 10-1, 10-5
using interactively 10-11
list type 10-5
changing 10-5
listing output from SQL precompiler 9-2
LISTTYPE
interactive keyword 10-16
lock level 11-4
LOCK statement 11-2

M

MARGIN keyword
PL/I 7-3
margins
in a COBOL program 5-3
PL/I 7-3
MARGINS keyword 9-15
use as precompiler input 9-2
member name 9-13
members
source file
changed 9-3
modified 9-3
temporary 9-3
messages 10-7, C-1
during a COBOL compile 9-17
during a PL/I compile 9-17
error and warning
ID C-1
text storage C-1
multiple predicates 3-10
multiple row insertion, notes 3-34

N

NAMING
interactive keyword 10-15
naming conventions 1-2
in a PL/I program 7-4
qualifying 10-2, 10-5
SQL naming 10-2
system naming 10-2
NFYOBJ parameter 11-4
NOT keyword 3-10
multiple search condition 3-27
NOT NULL 2-1, 3-35
NOT NULL WITH DEFAULT 3-35
notes on using a view 2-10
numeric
assignment 4-3
columns, rules for 3-18
conversions 13-2
data type 1-4

O

objects
catalogs 1-5
columns 1-3
compiled application program objects 1-10, 1-11
program 1-11
temporary source file member 1-11
user source file member 1-10
data bases 1-3
data dictionary 1-3
definitions 1-3
indexes 1-5
journal 1-3
journal receiver 1-3
rows 1-3
SQL 1-3
tables 1-3
views 1-4
OCCURS clause 5-7
online
help 10-6
index v
information v
OPEN CURSOR statement 3-23, 8-16
using with DESCRIPTOR clause 8-16
open cursors 3-25
OPEN statement 13-6
blocking considerations 9-19
performance considerations 13-6
use of 13-6
operators
comparison 3-9

OPTION keyword 9-13
options
See also keyword
SQL precompiler commands 9-12
OR keyword
multiple search condition 3-27
ORDER BY clause
used when selecting rows 3-14
used with DESC and ASC values 3-14
ordering rows 3-14
output
device selection 10-4
from precompiler 9-2
precompiler 9-2, 9-3
listing 9-2
printing current session 10-5
OUTPUT, PL/I element type 7-8
overflow
COBOL 5-7
PL/I 7-7
override considerations 9-19
overview, interactive SQL
audience 10-3

P

paging
through data, previously retrieved 13-7
through retrieved data 13-7
parameter
description 9-12
keyword
COMMIT 9-15
GENLVL (generation level) 9-15
INCFILE (include file) 9-14
MARGINS 9-15
OPTION 9-13
PRTFILE 9-15
SRCMBR (source member) 9-13
TEXT 9-16
PGM (program) keyword 9-12
program name 9-12
source file 9-13
SQL precompiler commands 9-12
SRCFILE 9-13
STRSQL 10-15
values
*ALL 9-15
*APOST (apostrophe) 9-14
*APOSTSQL 9-14
*CHG (change) 9-15
*COMMA 9-14
*CURLIB 9-12, 9-14, 9-15
*GEN 9-13
*LIBL 9-14, 9-15
*NOGEN 9-13
*NONE 9-15
*NOSOURCE 9-13
*NOSRC 9-13
parameter *(continued)*
values *(continued)*
*NOXREF 9-13
*PERIOD 9-14
*PGM 9-13
*QUOTE 9-14
*QUOTESQL 9-13
*SOURCE 9-13
*SQL 9-14
*SRC 9-13
*SRCFILE (source file) 9-14, 9-15
*SRCMBRTXT (source member text) 9-16
*SYS 9-14
*SYSVAL 9-14
*XREF 9-13
left right 9-15
library name 9-12, 9-14, 9-15
member name 9-13
printer file name 9-16
QLBLSRC 9-13
QPLISRC 9-13
QRPGSRC 9-13
QSYSPRT 9-16
QxxxSRC 9-13
severity level 9-15
source file name 9-13, 9-14
parameter markers 8-15
replacing with host variables 8-16
requirements for using 8-16
using 8-15
PEMPL in a COBOL program 5-4
performance considerations 13-2, 13-5
OPEN statement 13-6
PGM 9-12
PL/I programs 7-1
%INCLUDE directive 7-8
abbreviations 7-7
allowable declarations for host variables 7-5
application requirements 7-1
assignment rules 7-5
coding requirements 7-2
comments 7-2
communication area (SQLCA) 7-1
continuation for SQL statements 7-2
data types 7-7
declare restrictions 7-7
descriptor area (SQLDA) 7-1
dynamic SQL 7-1
element descriptions 7-8
error and warning messages 9-17
during a compile 9-17
EXECUTE IMMEDIATE statement 7-2
external file descriptions 7-8
host structures 7-3
host variables 7-3
margins 7-3
overflow 7-7
requirements for host variables 7-4

PL/I programs *(continued)*
reserved words 7-3
return code handling 7-11
sample program with SQL statements D-11, D-17
SQL statement delimiters 7-2
SQL syntax in 7-2
structures 7-9
truncation 7-7
using SQL statements 7-1
PL/I source file parameter (QPLISRC) 9-13
positioning at end of table 13-9
precompiler
basic processes 9-1
commands 9-12
keywords 9-12
parameter definitions 9-12
values 9-12
CRTSQLCBL 9-9
CRTSQLPLI 9-10
CRTSQLRPG 9-11
for COBOL 9-9
for PL/I 9-10
for RPG 9-11
INCLUDE statement 9-2
input to 9-2
options 9-12
output 9-3
output from 9-2
parameter definitions
*CURLIB 9-12
library name 9-12
PGM 9-12
program name 9-12
QLBLSRC 9-13
QPLISRC 9-13
QRPGSRC 9-13
QxxxSRC 9-13
source file name 9-13
SRCFILE 9-13
SQL commands 9-8, 9-9, 9-10, 9-11
temporary source file members 9-3
use of margins 9-2
using double-byte character set (DBCS) 9-2
precompiling SQL statements
precompile commands 9-8
predicates 3-8, 3-27
in a search condition 3-9
in the WHERE clause 3-8
multiple 3-10
use of 3-8
used in multiple search condition 3-27
prefix name
used with PL/I 7-9
PREPARE statement
using dynamically 8-4
preparing a program containing SQL for running 9-1
print current session 10-5
printer file (PRTFILE) 9-15
printer file name 9-16
problem solving
techniques for data base problems 13-7
procedure division (COBOL) 5-2
PROCESS
interactive keyword 10-15
program 1-11
description 1-11
name description 9-12
references 9-18
sample in COBOL D-3
sample in PL/I D-11
sample in RPG D-17
protection of data 11-1
PRTFILE 9-15
public authority 11-1

Q

QLBLSRC 9-13
QPLISRC 9-13
QRPGSRC 9-13
QSYSPRT 9-16
qualifying
SQL and system names 10-2
SQL statements 3-8

R

real data type 1-4
RECORD, PL/I element type 7-8
Recover SQL Session display
use of 10-7
recovery 10-7, 11-3
a failed SQL session 10-7
redundant information, providing 13-5
REFRESH
interactive keyword 10-16
register
USER special 3-10
relational data base, description 1-2
removing all entries from the current session 10-5
requirements for host variables
indicator variables and structure 7-7
reserved words
COBOL 5-3
PL/I 7-3
RPG 6-3
results table 3-20, 13-8
retrieving
a row from a set of rows 3-23
data a second time 13-7
data from the middle 13-8
from the beginning 13-7
order of rows in the second results table 13-8
in reverse order 13-9
information from a single table 2-4

return codes C-1
exception condition handling 4-7
handling by COBOL 5-8
handling by PL/I 7-11
handling for RPG 6-7
rules for handling 4-6
revising data 3-3
Revoke Object Authority (RVKOBJAUT) command 11-1
REVOKE statement 11-1
ROLLBACK statement
HOLD value 11-4
rows 1-3
arranging 3-14
combining 3-31
creating 2-2
current update 3-24
delete current 3-24
description 1-3
eliminating duplicate 3-25
inserting 2-2
inserting multiple rows 3-34
notes on 3-34
joining 3-31
ordering 3-14
retrieving a row from a set of rows 3-23
specifying a condition for a group of rows 3-12
rows in the second results table, order of 13-8
RPG programs 6-1
allowable declarations for host variables 6-5
application requirements 6-1
assignment rules 6-6
calculation section 6-2
coding requirements 6-2
comments 6-2
communication area (SQLCA) 6-1
continuation for SQL statements 6-2
descriptor area (SQLDA) 6-3
dynamic SQL 6-3
example of SQL statements in RPG 6-2
external descriptions 6-6
host structures 6-4
host variables 6-3
indicator variables 6-6
requirements for host variables 6-5
reserved words 6-3
return code handling 6-7
SQL syntax in 6-2
structures 6-6
using SQL statements 6-1
RPG source file parameter (QRPGSRC) 9-13
rules
assignment rules 4-2
for character string assignment 4-3
for coding host variables in COBOL 5-4
for coding host variables in RPG 6-5
for numeric assignment 4-3
for PL/I host variables 7-4
rules *(continued)*
for SQL with host language 4-6
for SQL with host languages 4-2
host structures 4-4
relating to examples of SQL statements iii
return code handling 4-6
SQL with host languages 4-4
using SQL with host languages 4-1
running a program containing SQL statements
diagnostic information 9-1
RVKOBJAUT (Revoke Object Authority) command 11-1

S

sample programs D-1
save/restore 11-5
saving session in source file 10-5
search conditions 3-26
using predicates 3-9
security
data 11-1
SQL objects 11-1
using views 11-1
SELECT INTO statement 3-5
SELECT statements 3-5, 8-5
allocating storage 8-10
combining two or more statements 3-16
descriptor area (SQLDA) 8-7
DISTINCT keyword 3-25
dynamic processing 8-3
dynamic SQL 8-3
fixed-list 8-5
processing dynamically 8-5
selecting several rows at a time 3-5
using a cursor 8-14
varying-list 8-5, 8-6
with UNION keyword 3-16
selecting
all the columns 3-7
data from more than one table 3-29
data from multiple tables 13-5
improving performance 13-5
joining tables 13-5
provide redundant information 13-5
one row 3-5
output device 10-4
rows from a table 3-5
sequence numbers
in COBOL 5-3
in RPG 6-3
session history 10-4
session services 10-4
changing data base (library) lists 10-4
changing list types 10-5
commitment control 10-4
controlling SELECT output device 10-4
printing current session 10-5

session services *(continued)*
removing entries from current session 10-5
saving sessions in source file 10-5
statement processing control 10-4
session services function 10-1
sessions 10-7
changing options 10-1
controlling 10-1
functional description 10-7
printing current 10-5
recovery of a failed session 10-7
removing all entries from the current session 10-5
saving in source file 10-5
session services function 10-1
SET clause 3-3
severity level 9-15
shift-in, shift-out characters
use with interactive SQL 10-14
small integer (SMALLINT) data type 1-4
SMALLINT
SMALLINT data type 1-4
source commands, SQL precompiler 9-8
source file (SRCFILE) 9-13, 9-15
members 9-3
changed 9-3
modified 9-3
temporary 9-3
name 9-13, 9-14
source file parameters
QLBLSRC (COBOL) 9-13
QPLISRC (PL/I) 9-13
QRPGSRC (RPG) 9-13
source member (SRCMBR) 9-13
special register 3-10
specifying the columns you want 3-7
SQL descriptor area (SQLDA)
description 8-7
SQL in RPG example 6-2
SQL index, using effectively 13-2
avoid LIKE predicates beginning with % or _. 13-3
avoid numeric conversions 13-2
avoid string character padding 13-2
conditions with out an index 13-3
SQL naming convention 10-2
SQL precompiler
basic processes 9-1
SQL statements iii
rules for examples of iii
used in sample COBOL program D-3
used in sample PL/I program D-11
used in sample RPG program D-17
SQLCA
COBOL 5-1
PL/I 7-1
RPG 6-1
SQLCODEs C-1
description C-1
how to display descriptions C-1
SQLCODEs *(continued)*
list C-1
message file C-1
occurrence of 3-6
SQLD 8-8
SQLDA
COBOL 5-3
description 8-7
example of allocating storage 8-10
format description 8-7, 8-8, 8-9
SQLD 8-8
SQLDABC 8-7
SQLDAID 8-7
SQLDATA 8-8
SQLIND 8-9
SQLLEN 8-8
SQLN 8-7
SQLNAME 8-9
SQLRES 8-8
SQLTYPE 8-8
SQLVAR 8-8
PL/I 7-1
RPG 6-3
SQLDABC 8-7
SQLDAID 8-7
SQLDATA 8-8
SQLIND 8-9
SQLLEN 8-8
SQLN 8-7
SQLNAME 8-9
SQLRES 8-8
SQLTYPE 8-8
SQLVAR 8-8
SRCFILE (source file) 9-13
SRCMBR (source member)
description 9-13
Start Debug (STRDBG) command 12-2
Start SQL (STRSQL) command 10-15
statement entry function 10-1, 10-3
statement processing
controlling 10-4
statements in RPG programs
SQL in RPG example 6-2
statements, SQL 3-1
used in sample COBOL program D-3
used in sample PL/I program D-11
used in sample RPG program D-17
static SQL statements 8-2
STRCMTCTL 11-4
STRDBG (Start Debug) CL command 12-2
string, character assignment 4-3
STRSQL (Start SQL) command
description 10-15
parameters 10-15
syntax 10-15
structures
COBOL 5-3, 5-8
halfword integer variables 4-4

structures *(continued)*
indicator 4-4
PL/I 7-7, 7-9
RPG 6-4, 6-6
RPG data 6-5
supported SQL statements 10-8
syntax checking
selecting interactively 10-4
syntax for SQL precompiler commands 9-8
CRTSQLCBL 9-9
CRTSQLPLI 9-10
CRTSQLRPG 9-11
SYSCOLUMNS 1-6
SYSINDEXES 1-8
SYSKEYS 1-8
SYSTABLES 1-9
system naming convention 10-2
system print file, QSYSPRT parameter 9-16
SYSVIEW 1-10
SYSVIEWDEP 1-9

T

table definitions 13-10
tables 1-3, 2-1
changing information in a table 2-6
CREATE TABLE statement 2-2
creating 2-2
creating tables 2-1
defining table names 2-2
deleting information in a table 2-7
description 1-3
get information from a single table 2-4
getting information from more than one table 2-5
inserting information into tables 2-2, B-3
join 3-33
updating 2-6
using tables 2-1
working with 2-1, 2-2, 2-4, 2-7
tables, views, and catalogs
using COMMENT ON 2-13
TDEPT (department table)
creating 2-2
techniques
coding 3-1
techniques for solving common data base problems 13-7
adding to the end of a table 13-9
changing table definitions 13-10
keeping a copy of the data 13-7
positioning at end of table 13-9
retrieving the data a second time 13-7
updating data as it is retrieved from the data base 13-9
updating data previously retrieved 13-10
techniques for using SQL 13-1
selecting data from multiple tables, improving performance 13-5
statements 13-1
temporary source file members 1-11, 9-3
terminology 1-2
data base 1-2
qualifying 10-2
SQL naming 10-2
SQL objects 1-2
system 1-2
system naming 10-2
test data structures 12-1
designing 12-1
test tables 12-2
test data structures, designing
test views of existing tables 12-1
test tables 12-2
test views of existing tables 12-1
testing 12-1
authorization 12-1, 12-2
GRANT statement example 12-2
debugging your program 12-2
establishing a test environment 12-1
SQL statements 12-1
SQL statements interactively 10-14
test data structure 12-1
test data structures 12-1
test input data 12-1
test tables 12-2
TEXT 9-16
tips on using interactive SQL 10-11
truncation
COBOL 5-7
PL/I 7-7

U

UNION ALL 3-19
keeping duplicates 3-19
UNION keyword 3-16
length and data type rules 3-18
rules for using 3-17, 3-18
unit of recovery
completion of 3-25
description 3-25
UPDATE statement 3-3, 13-9
updating the current row 3-24
using 2-6
WHERE CURRENT OF clause 3-24
updating data
previously retrieved 13-10
retrieved from a data base 13-9
restrictions 13-9
updating the current row of a set of rows 3-24
user source file member 1-10
USER special register 3-10
USER1 data base
creating 2-1
using
a view 2-8
catalogs in data base design 2-11

using *(continued)*
tables 2-1
using a cursor 8-14
using basic SQL statements and clauses 3-1
using interactive 10-3
using SQL statements 13-1
in COBOL programs D-3
in PL/I programs D-11
in RPG programs D-17
using an SQL index effectively 13-2

V

validity checking
selecting interactively 10-4
value 9-12
value item (COBOL) 5-7
VALUES clause 3-1
values, inserting 3-35
variables 4-1
address variables used in dynamic SQL 8-1
assignment rules for COBOL 5-5
assignment rules for PL/I 7-5
assignment rules for RPG 6-6
COBOL 5-3
COBOL declarations 5-5
COBOL indicator 5-8
halfword integer 4-4
indicator 4-4
input host 8-15
PL/I 7-3
PL/I declarations 7-5
PL/I indicator 7-7
replacing parameter markers with host 8-16
requirements for COBOL 5-4
requirements for PL/I 7-4
requirements for RPG 6-5
RPG 6-3
RPG declarations 6-5
RPG indicator 6-6
RPG names 6-1
using host 4-1
using parameter markers 8-15
varying-length character string
COBOL 5-7
length item 5-7
varying-list
SELECT statement 8-6
used with dynamic SQL 8-5
views 1-4, 2-1, 3-8
authority to 11-1
catalog
SYSCOLUMNS 1-6
SYSINDEXES 1-8
SYSKEYS 1-8
SYSTABLES 1-9
SYSVIEWDEP 1-9
SYSVIEWS 1-10
views *(continued)*
changing 2-10
changing information in a table 2-8
creating a view 2-8
defining 2-10
get information from a single view 2-4
getting information from 2-10
inserting information 2-10
processing 2-10, 3-8
running 2-10
updating 2-10
using a view 2-8
working with 2-1, 2-4

W

warning and error messages
during a COBOL compile 9-17
during a PL/I compile 9-17
WHENEVER NOT FOUND clause 3-23
WHENEVER statement 4-7, 6-7
exception condition 4-7
handling 4-7
handling by RPG 6-7
WHERE clause 3-8, 3-27, 3-31, 13-4
AND 3-27
multiple search condition 3-27
NOT 3-27
OR 3-27
used with a join 3-31
used with host variables 4-1
using 2-6
WHERE CURRENT OF clause 3-24
working with
data bases 2-1
indexes 2-11

AS/400™ Programming:
Structured Query Language/400
Programmer's Guide

SC21-9609-0

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your IBM-approved remarketer. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

☐ If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your IBM representative or your IBM-approved remarketer to request additional publications.

Name _____

Company or Organization _____

Address _____

_____

City State Zip Code

Phone No. _____

Area Code

No postage necessary if mailed in the U.S.A.

Fold and Tape **Please do not staple**

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS / PERMIT NO. 40 /ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Development
Department 245
Rochester, Minnesota, U.S.A. 55901

Fold and Tape **Please do not staple**

IBM®

AS/400™ Programming:
Structured Query Language/400
Programmer's Guide

SC21-9609-0

READER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Direct any requests for additional publications, technical questions about IBM systems, changes in IBM programming support, and so on, to your IBM representative or to your IBM-approved remarketer. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

☐ If your comment does not need a reply (for example, pointing out a typing error), check this box and do not include your name and address below. If your comment is applicable, we will include it in the next revision of the manual.

☐ If you would like a reply, check this box. Be sure to print your name and address below.

Page number(s): Comment(s):

Please contact your IBM representative or your IBM-approved remarketer to request additional publications.

Name _____

Company or Organization _____

Address _____

_____

City State Zip Code

Phone No. _____

Area Code

No postage necessary if mailed in the U.S.A.

Fold and Tape **Please do not staple**

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS / PERMIT NO. 40 /ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Development
Department 245
Rochester, Minnesota, U.S.A. 55901

Fold and Tape **Please do not staple**

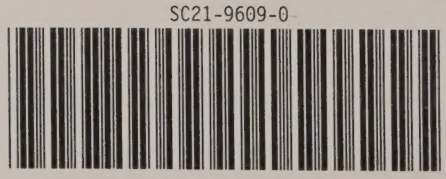
IBM®

Cut Along Line

Cut Along Line

IBM®

Program Number
5728-ST1



Printed in U.S.A.